

Navigating the Semantic Web with Logical Information Systems

Sébastien Ferré^{*}
ferre@irisa.fr

Abstract: Exploratory search is about browsing and understanding an information base. It requires a more interactive process than retrieval search, where the user sends a query and the system returns answers. Logical Information Systems (LIS) support exploratory search by guiding users in the construction of queries, and by giving summaries of query results. The contribution of this paper is to adapt and extend LIS to the Semantic Web. We define a summarization and navigation data structure that provides rich views over data, and guides users from view to view. This guiding is proved consistent (i.e., never leads to an empty result set), and complete (i.e., every query can be reached by navigation only). The query language covers a large fragment of SPARQL, and is more concise by using a syntax close to description logics. Our approach could be implemented on top of existing tools for storing, reasoning and querying.

Key-words: semantic web, information retrieval, navigation, logical information system

Naviguer dans le web sémantique avec les systèmes d'information logiques

Résumé : *La recherche d'information exploratoire concerne le parcours et la compréhension d'une base d'information. Elle requiert un processus plus interactif que la recherche d'information classique, où l'utilisateur émet une requête et le système retourne des réponses. Les Systèmes d'Information Logiques (LIS) supporte la recherche exploratoire en guidant les utilisateurs dans la construction de requêtes et en produisant des résumés des réponses à la requête. La contribution de cet article est l'adaptation et l'extension des LIS pour le web sémantique. Nous définissons une structure de données de résumé et de navigation qui fournit des vues riches sur les données, et guide les utilisateurs de vue en vue. Ce guidage est prouvé cohérent (c-à-d., ne mène jamais à un résultat vide) et complet (c-à-d., permet d'atteindre toutes les requêtes par simple navigation). Le langage de requêtes couvre un large fragment de SPARQL, et est plus concis en utilisant une syntaxe proche de celle des logiques de description. Notre approche pourrait être implémentée au dessus d'outils existant pour le stockage, le raisonnement et l'interrogation.*

Mots clés : *web sémantique, recherche d'information, navigation, systèmes d'information logiques*

^{*} Équipe LIS, IRISA, Université de Rennes 1 (travail réalisé avec le soutien d'une délégation CNRS)

1 Introduction

With the growing amount of available resources in the Semantic Web, it is a key issue to provide an easy and effective access to data and knowledge, not only to the specialist, but also to the casual user. The challenge is not only to allow users to retrieve particular resources (e.g., flights), but to support them in the exploration of a domain knowledge (e.g., which are the destinations? which are the most frequent? with which companies and at which price?). We call the first mode *retrieval search*, and the second mode *exploratory search* [Mar06, ST09].

We claim that most existing querying languages for the Semantic Web (e.g., SPARQL [PAG06], SPARQL-DL [SP07], OWL-QL [FHH04]), while expressive, are difficult to use, even for specialists, and do not provide enough feedback to satisfy exploratory search. Even if a user has a perfect knowledge of the syntax and semantics of the query language, she may be ignorant about the application vocabulary. Therefore, the class and property names she would use would certainly be undefined. A first approach is to use a *semantic distance* [CKZ04] between classes and properties in order to match the query to resources. The problem with any distance is that it makes hidden assumptions on the user intent. A second approach is to use a query assistant (e.g., Protégé) that suggests relevant classes and properties given a detailed modelling (e.g., domain and range of properties). In fact, this modelling is neither sufficient, nor entirely necessary. It is not sufficient because a query that is syntactically correct and semantically consistent w.r.t. the model can still produce no result (e.g., there might be a flight from Rennes to Washington, but it happens there is none). Some part of the modelling may be useful for inference purposes, but we show in this paper that it is not necessary for guiding users in the composition of useful queries.

Logical Information Systems (LIS) [FR04] have been introduced to solve the dilemma between an expressive querying language that is difficult to use (e.g., Boolean queries), and an intuitive but rigid navigation structure (e.g., file hierarchies). The solution retains the use of an expressive query language, but frees users from directly composing and modifying queries. Instead, a navigation process takes place, where the system provides suggestions to modify the query (navigation links), and the user makes choices. This navigation process has the following properties: (1) the user has full control on the search, and can make choices in any order, (2) the system makes no assumption on the user intents, (3) every information can be accessed through every possible path, (4) the user is guaranteed never to get an empty result (no dead-end). At each step of the navigation, the set of suggestions is organized as a summary of the current selection that provides insight and feedback on data, and thus supports exploratory search. LIS have been applied to real applications such as photo management [Fer09], and geographical information systems [BFRQ08].

The contribution of this paper is to adapt and extend the LIS approach to the Semantic Web. LIS initially apply to a small subset of RDFS (`rdf:type` and `rdfs:subClassOf` are the only properties). A first extension to the Semantic Web has been proposed [AF08], but with a query language restricted to the language of the basic description logic \mathcal{ALC} . We here propose a navigation process that is (1) based on a query language whose *expressivity* is similar to SPARQL, and (2) as *natural* as the notation N3, and that is (3) *consistent* (no dead-end) and (4) *complete* (every query can be reached by navigation). Our approach builds upon and is compatible with existing techniques for designing and storing ontologies, reasoning, as well as querying languages and their implementations. In fact, the LIS approach should be seen as a layer on top of querying mechanisms, rather than an alternative to it.

We first give a short introduction of the principles of LIS in Section 2, before detailing in Section 3 how these principles are applied to the Semantic Web. We introduce a new query language with a DL-based syntax, and SPARQL-based semantics. Its expressivity is discussed in Section 4. A few perspectives are discussed in Section 4 before concluding.

2 Logical Information Systems

Logical Information Systems (LIS) [FR04, Fer09] follows the logic-based approach to information retrieval [vR86]. A dataset is a set of objects, each object being described by a logical formula. On top of this, Formal Concept Analysis (FCA) [GW99], and more specifically its logical generalization [FR04], brings the rationale for tightly combining querying, navigation, and annotation. FCA defines a data structure, the *concept lattice*, that is automatically derived from the dataset, and is used for information retrieval, data analysis, and data mining. LIS use it as a summarization and navigation structure. Every change in the dataset automatically entails the update of the concept lattice, and hence of the summarization and navigation structure.

The LIS user interface is a *local view* of the concept lattice centered on a concept, the *focus*. The local view is made of three parts: (1) the query, (2) the extension, and (3) the index. The query is a logical formula. The extension is the set of objects that are matched by the query, along the logic-based approach [vR86]. The extension identifies the focus concept. Finally, the index is a finite subset of the logic that is restricted to formulas that match at least one object in the extension. The index plays the role of a summary or inventory of the extension, showing which kinds of objects there are, and how many of each kind there are. Figure 1 shows a screenshot of a LIS interface, where objects are photos, and formulas are about date, location, event, persons, etc.

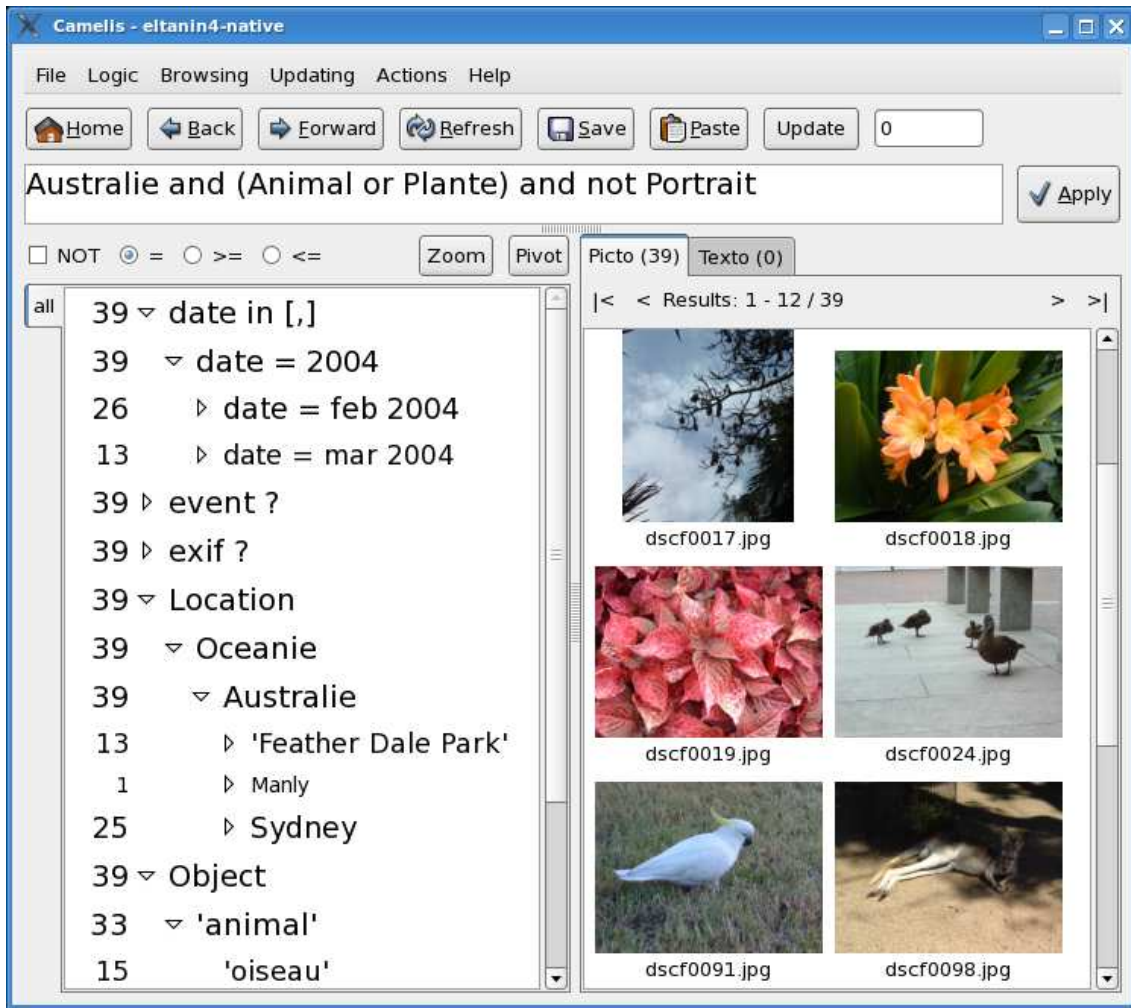


Figure 1: A screenshot of the user interface of CAMELIS, a LIS software.

The query can be modified in three ways. To *query by formula* is to directly edit the query, which requires expertise or luck from the user. To *navigate* is to select formulas in the index in order to make the query more specific or more general. To *query by examples* is to select a set of objects in the extension, which leads to the query that is the conjunction of all common properties of the selected objects. In the three cases, the modification of the query entails the update of the extension, hence of the focus concept and the index. By definition of the index, no navigation link (a selection in the index) can lead to an empty result. Conversely, because the navigation structure is a lattice rather than a hierarchy, all valid conjunctions of formulas can be reached by navigation, and in any order. Contrary to querying by formula, navigation only requires from the user to recognize the meaning of formulas in the index, in the context of the application.

There exist several implementations of LIS¹. Camelis [Fer09] (see Figure 1) is the most accessible to end-users with a dedicated graphical interface, and modules to extract metadata from photos, MP3 files, etc. It has been applied to the management of all sorts of files such as photos, music files, Java libraries, BibTeX files. LISFS [PR03] is a Linux file system, whose API is redefined along LIS principles. GEOLIS [BFRQ08] is a geographical information system based on LISFS. It has a web interface similar to the Camelis' interface except the extension is rendered on a map.

3 Navigating the Semantic Web

The contribution of this paper is to adapt the LIS framework to the Semantic Web. We consider the dataset to be an RDF graph [PAG06], which may include both explicit and implicit facts (i.e., triples) through reasoning. In this paper, we do not make the distinction between the two. In practice, the implicit facts may be materialized in a preprocessing stage, or inferred

¹see <http://www.irisa.fr/LIS/software>s

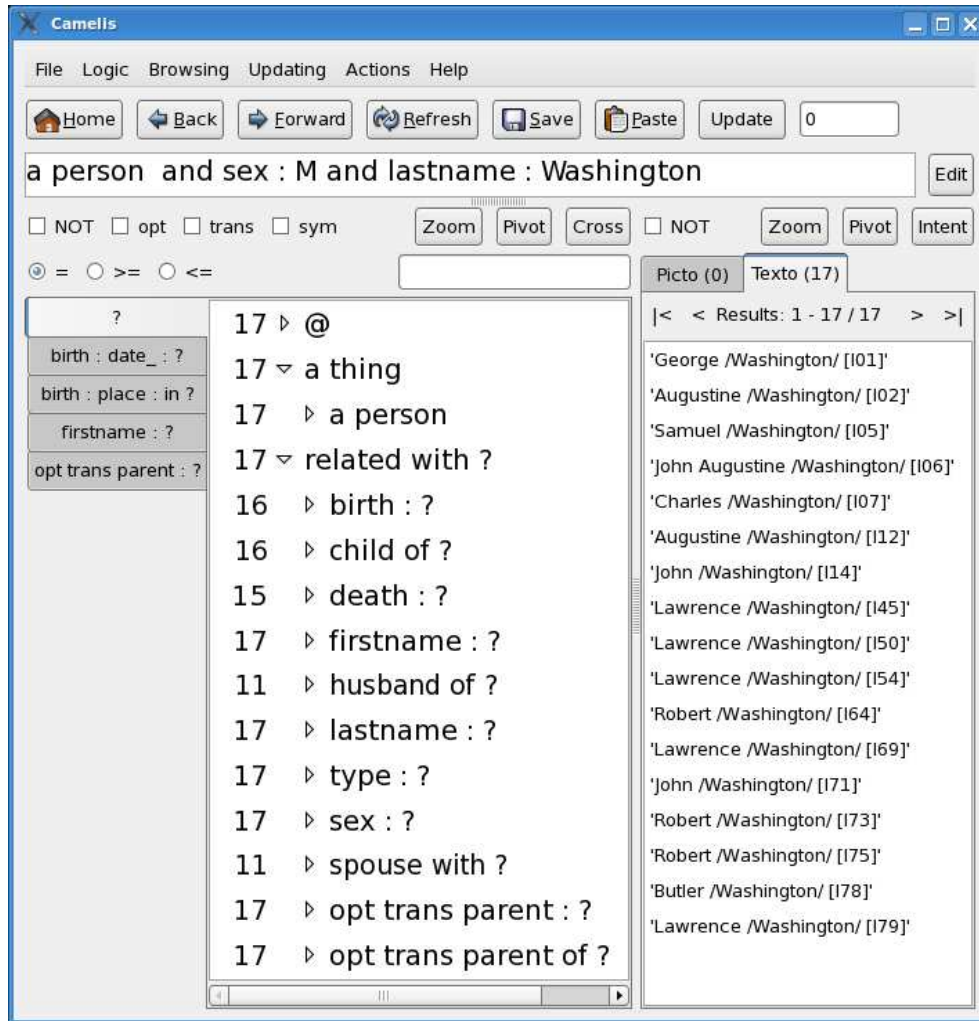


Figure 2: A local view: query (top), extension (right), and index (left).

on demand. Inference rules can be OWL axioms, or SWRL rules. We assume pairwise disjoint infinite sets of IRIs (I), blank nodes (B), and literals (L). We call *resources* the set $R = I \cup B \cup L$ resulting from the union of IRIs, blank nodes, and literals (indeed, RDFS defines `rdfs:Literal` as a subclass of `rdfs:Resource`).

Definition 1 (RDF graph) An RDF graph is defined as a set of triples $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$, where s is the subject, p is the predicate, and o is the object.

For illustration purposes, we consider RDF graphs about genealogical data. The IRIs of this domain are associated to a namespace `gen:.` This prefix is omitted if there is no ambiguity. Resources can be *persons*, *events*, *places* or literals such as names or dates. Persons belong to the class of *men* or *women*, may have a *firstname*, a *lastname*, a *sex*, a *father*, a *mother*, a *spouse*, and a *birth*. A birth is an event that may have a *date* and a *place*. Places can be described as *parts* of larger places. The use of OWL axioms may help to enforce some invariants, e.g., the spouse property is symmetrical, the father property is functional, and the part property is transitive.

In Section 3.1, we first redefine the LIS notion of *local view*. This comprises the definition of queries, the extension of queries, and the summarization index over extensions. Then, navigation links are defined in Section 3.2 on top of the local view. They allow users to navigate between different local views in a flexible way, and without having to type anything. A set of navigation links that has a practical size is proved consistent and complete. Figure 2 shows the user interface of our prototype, applied to the genealogy of George Washington (see <http://www.irisa.fr/LIS/ferre/camelis/camelis2.html>). It reflects the structure of a local view with the query at the top, the extension at the right, and the index at the left. This paper does not focus on interfaces, and this prototype has experimentation and illustration purposes for our navigation model.

3.1 Local View

3.1.1 Queries and Extensions

In LIS, the *query* defines the *focus* of the local view. For instance, a local view may put the focus on “women whose some parent was born in Virginia in 1642”. This focus can be defined by the SPARQL query below. SPARQL is a W3C recommendation for querying RDF datasets. It is based on graph patterns that are matched over RDF graphs.

```
SELECT ?x
WHERE { ?x rdf:type gen:woman .
        ?x gen:parent ?p . ?p gen:birth ?b .
        ?b gen:date 1642 .
        ?b gen:place ?l . gen:VA gen:part ?l}
```

A LIS query must necessarily define a set of resources, i.e., a mono-dimensional relation. This implies that we only need SPARQL queries with a single variable in the SELECT clause. This also means our queries are analogous to OWL complex classes (a.k.a. concept descriptions in Description Logics [BCM⁺03]). In fact, the above query can be expressed in the description logic *SHOIN* that is behind OWL-DL:

$$Woman \sqcap \exists parent.\exists birth.(\exists date.\{1642\} \sqcap \exists place.\exists partOf.\{FRANCE\}).$$

The property *partOf* is the inverse of *part*. The advantages of the DL syntax is that it is much more concise, and avoids the use of variables. LIS do not require from end-users the ability to write queries, but they do require from them to understand queries. Furthermore, we do not expect end-users to have education in computer science, and ideally, the queries should be understandable with little, if any, learning. We think the DL syntax is closer to this objective than the SPARQL syntax. At the same time, SPARQL has more expressive patterns (e.g., cycles), and a semantics that is less demanding w.r.t. logical inference.

We propose a new language for querying RDF graphs, where query results are restricted to sets of resources. Therefore, the expressions of this language are called *complex classes*. It also makes use of *complex properties*, derived from basic properties. This language is given a semantics by translating complex classes to SPARQL queries, and its expressiveness is compared to SPARQL in Section 4.

Definition 2 (complex property) A complex property is any of the expressions:

- p* : the property *p* itself,
- p* of the inverse of the property *p*,
- p* with the symmetric closure of the property *p*,
- trans *P* the transitive closure of the complex property *P* (“transitively *P*”),
- opt *P* the reflexive closure of the complex property *P* (“optionally *P*”).

OWL allows to declare a name for the inverse of a property, e.g., **child** is the inverse of **parent**. In this case, we have that **child** : and **parent** of are equivalent. An advantage of our language is that it allows to read all properties in both directions, whether a name for the inverse has been defined or not. Similarly, OWL allows to declare a property as symmetric or transitive, and our language allows to use such closures on-the-fly. For instance, the complex property **parent** with allows to reach parents and children at the same time, whereas it is unlikely to have a property name defined to this purpose. If a property **ancestor** is defined, then it is equivalent to **trans parent**. In the following, we use **in** as an abbreviation for the complex property **opt trans part of**. Applying the three closures, **opt trans *p* with**, defines an equivalence relation, while **opt trans *P*** defines a partial ordering if *P* is antisymmetric, and a pre-order otherwise.

Definition 3 (complex class) Let *V* be an infinite set of variables, disjoint with the set of resources *R*. For every resource $r \in R$, variable $v \in V$, IRI *i*, complex property *P*, and complex classes *C*, *C*₁, *C*₂, the following expressions are also complex classes:

$$r, ?v, ?, P C, a i, C_1 \text{ and } C_2.$$

Compared to DL languages, the complex class *r* corresponds to a *nominal* or *enumerated class* containing exactly one individual, the anonymous variable ? corresponds to the *top concept* \top , the expression *P C* corresponds to a *qualified existential restriction* $\exists P.C$ (simply a *restriction* from now on), the expression **a *i*** corresponds to a *concept name*, and the **and** corresponds to *concept intersection* \sqcap . The addition of variables allows the expression of cyclical graph patterns, like in SPARQL. The notation *p* : is reminiscent of the notation of valued attributes. For example, in the expression **name** : “John”, **name** is the attribute, and “John” is the value. The expression can be read “has name John”, or “whose name is John”. The above query can now be written:

a woman and parent : birth : (date : 1642 and place : in VA)

A semantics for our language, and a practical way to compute answers to queries in this language, is obtained by defining a translation to SPARQL.

Definition 4 Let C be a complex class. The SPARQL translation of C is defined by

$$\Gamma(C) = \text{SELECT } ?x \text{ WHERE } \Gamma_x(C)$$

where $x \in V$ is a variable not occurring in C , and where Γ_x is defined by induction on complex classes: for every complex class C, C_1, C_2 , and complex property P ,

- $\Gamma_x(r) = \text{FILTER } \{ ?x = r \}$, where $r \in R$ is a resource,
- $\Gamma_x(?v) = \text{FILTER } \{ ?x = ?v \}$, where $v \in V$ is a variable,
- $\Gamma_x(?) = \{ \}$ (the empty graph pattern),
- $\Gamma_x(P C) = \{ \{ ?x P ?y \} \Gamma_y(C) \}$, where y is a fresh variable,
- $\Gamma_x(\mathbf{a} \ i) = \{ ?x \text{ rdf:type } i \}$, where $i \in I$ is a class IRI,
- $\Gamma_x(C_1 \ \text{and} \ C_2) = \{ \Gamma_x(C_1) \ \Gamma_x(C_2) \}$.

The triples whose predicate is not a basic property can be handled by defining a new property with OWL axioms or deductive rules. It is also possible to rewrite those triples into SPARQL graph patterns, except for transitivity, which requires regular expressions as predicates (nSPARQL [PAG08]):

- $?x \ p \ \text{of} \ ?y \rightarrow ?y \ p \ ?x$,
- $?x \ p \ \text{with} \ ?y \rightarrow \{ ?x \ p \ ?y \} \ \text{UNION} \ \{ ?y \ p \ ?x \}$,
- $?x \ \text{opt} \ P \ ?y \rightarrow \{ ?x \ P \ ?y \} \ \text{UNION} \ \{ \text{FILTER } \{ ?x = ?y \} \}$.

We can now define the second part of a local view, the extension. It is simply defined as the answers to the SPARQL translation of the query.

Definition 5 (extension) Let C be a complex class. The extension of C , noted $\text{ext}(C)$, is the set of resources that are answers to the SPARQL translation of C . Every element of the extension is called an instance of the complex class C .

Given the constraint of a single distinguished variable in SPARQL queries, there may be more efficient ways to compute the extension, compared to generating the SPARQL query and computing its answers. However, the implementation details and optimizations are out of the scope of this paper.

3.1.2 Summarization Index

The third part of the local view is the *index* which serves as a summary of the extension. Every *index term* is a descriptor of some or all resources in the extension. Therefore, every index term can be seen as a refinement of the current query. Lemma 7 proves that, in most cases, the extension of the refined query can be reduced to an intersection.

Definition 6 (index term) Let q be a complex class representing the query of a local view. A complex class x is an index term of q , which we note $x \in \text{index}(q)$, if $\text{ext}(q \ \text{and} \ x) \neq \emptyset$.

Lemma 7 Let q, x be complex classes. Unless q and x have variables in common, we have $\text{ext}(q \ \text{and} \ x) = \text{ext}(q) \cap \text{ext}(x)$.

This lemma provides a valuable optimization because the query extension $\text{ext}(q)$ is already computed in the local view, and the extension of index terms can be cached for the most common of them. The number of index terms can be infinite, but in practice only a limited subset is presented to the user at any given time. Initially, a small index is presented, and then the user can expand it in a controlled way to see more index terms. A number of statistical indicators can be attached to an index term x to qualify its relation to the query q , e.g.:

- $\text{count}(x, q) = \|\text{ext}(q \ \text{and} \ x)\|$,
- $\text{precision}(x, q) = \frac{\|\text{ext}(q \ \text{and} \ x)\|}{\|\text{ext}(q)\|}$,

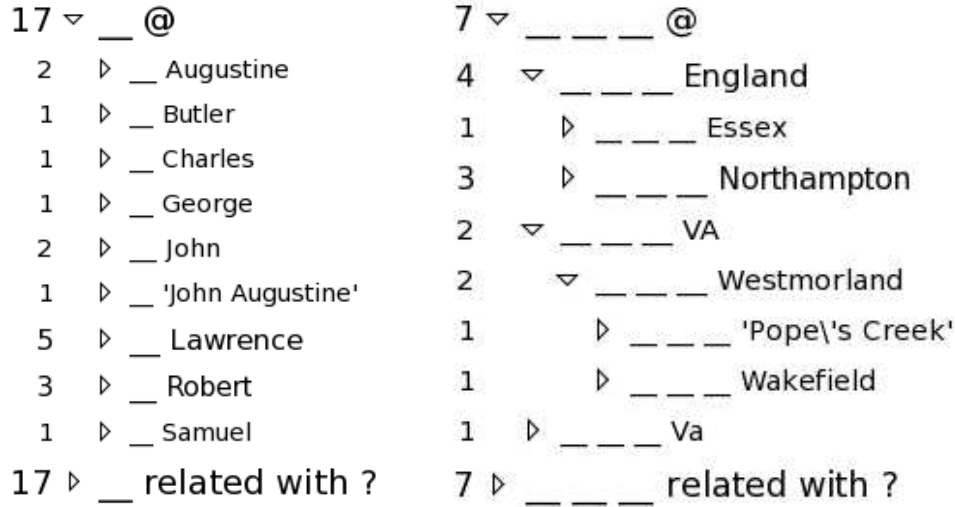


Figure 3: Index subtrees for the query a individual and sex : M and lastname : Washington: (a) under firstname : ?, (b) under birth : place : in ? (@ is used to group resource index terms, and ___ is used to avoid repetition of a same property).

- $recall(x, q) = \frac{\|ext(q \text{ and } x)\|}{\|ext(x)\|}$.

Instead of presenting the index terms as a flat list, they can be organized into a partial ordering \leq that reflects subsumption relationships between them. Figure 2 and 3 show how this partial ordering can be rendered as trees of complex classes. The number at the left of each index term is its count. Figure 3 gives the number of male Washington’s having a given firstname (Figure 3(a)), and being born in a given place (Figure 3(b)). It is not required that this partial ordering be complete w.r.t. subsumption because it does not affect query answering. To one end, we can just rely on the subsumption relationships between class names (`rdfs:subClassOf`) and property names (`rdfs:subPropertyOf`) as done in RDFS. On the other end, we can exploit the full reasoning capabilities of OWL-DL to infer subsumption relationships between complex classes and between complex properties. The guiding criteria to design this partial ordering is that it should be intuitive to users (i.e., they can anticipate the inferred subsumptions), it should provide enough structure to the index, and it should be of practical complexity.

In the illustrations of this paper, we use only RDFS inference through the properties `rdfs:subClassOf` and `rdfs:subPropertyOf`. In the partially ordered index, every class is placed under its superclasses. For instance, a woman \leq a person \leq a thing. Every property is placed under its superproperties, and also under its different closures. For instance, father of \leq parent of \leq trans parent of. From this ordering of complex properties, restrictions can also be ordered:

$$P_1 C_1 \leq P_2 C_2 \iff P_1 \leq P_2 \wedge C_1 \leq C_2. \tag{1}$$

For instance, father of a man \leq parent of a person. Every index term is placed under the anonymous variable ?, which then plays the role of the root of the whole index. Similarly, all restrictions in the form $P C$ are grouped under $P ?$.

We have noted above that a reflexive and transitive complex property, i.e., in the form `opt trans P`, is a partial ordering. It would be useful that the index reflects this partial ordering through its own partial ordering. This is already the case for the property `rdfs:subClassOf` (resp. `rdfs:subPropertyOf`) through the complex classes a i (resp. $P C$), as explained above. This can be generalized by observing that the following subsumption holds for every complex property P , and every IRIs i_1, i_2 :

$$\text{opt trans } P i_1 \leq \text{opt trans } P i_2 \iff i_1 \in ext(\text{opt trans } P i_2). \tag{2}$$

This is illustrated in Figure 3(b) with the birth place of male Washington’s: e.g., in Westmorland \leq in VA, because Westmorland $\in ext(in VA)$. In fact, the complex class a i is equivalent to `type : opt trans rdfs:subClassOf : i`, where i is a class IRI. Therefore, the index under this complex class reveals the same class subsumption hierarchy as under a thing. We now list a few other examples in the genealogy ontology:

- `place : in ?` is the root of a natural hierarchy of locations, from the larger locations to the smaller;
- `opt trans parent` : is the root of the descendancy chart of the ancestors of the selected people, showing under each individual its children, and this recursively;

- `opt transparent of` is the root of the upside-down ancestry chart of the descendants of the selected people, showing under each individual its parents, and this recursively.

3.2 Navigation

A local view is determined by its query. The query determines the extension and the index as presented above. By default, the user is initially presented with the local view of the most general query `?`, whose extension is the set of all resources defined in the dataset. Alternately, an application can be parameterized by a more specific *home* query: e.g., the query `a person` in a genealogical dataset because the focus is then on persons. In their search for information, users need to change the focus, i.e., to change the current query. In retrieval search, users are looking for a particular set of resources, and try and find the query whose extension matches this set of resources. For instance, to answer the question “Which women are married to a Washington?”, we can use the query `a woman and spouse with lastname : Washington`. In exploratory search, users are looking for patterns in data rather than for particular resources. For instance, if the user is interested in the birth place of people having lastname “Washington”, she can set the query to `lastname : Washington`, and then explore the index under the index term `birth : place : in ?`. She obtains a natural hierarchy of places, where each place is annotated by the proportion of the Washington’s that were born in this place. The index also informs about their birth dates, ancestors, descendants, etc.

A well informed user can of course directly type in queries. However, as explained in the introduction, this requires not only to have good knowledge of the query language (syntax and semantics), and of the domain-specific vocabulary (e.g., `man`, `place`), but also of the contents of the dataset if one wants to avoid empty results. This is paradoxical as the less we know a dataset, the more we need to search it. We propose to define navigation links between local views, so as to guide users. We prove in this section that this guiding is both *consistent*, i.e., never leads to empty results, and *complete*, i.e., can lead to arbitrary local views. The index plays a central role in navigation, in addition to summarization. It is really what makes LIS different to other approaches.

There are only three primitive kinds of navigation links: *zoom-in*, *naming*, and *reversal*. The zoom-in applies to an index term, the naming applies to a variable name (generated or user-given), and the reversal applies to a syntactic element of the query. If we see the query in its SPARQL form, the zoom-in extends the graph pattern, while the reversal changes the variable in the SELECT clause, thus changing the point of view. The naming makes a variable of the SPARQL query visible in the LIS query.

Definition 8 (navigation link) *A navigation link l is a function that takes a complex class as input (the initial query q), and returns a new complex class as output (the resulting query q'): $l(q) = q'$. For convenience, we note $q [l] q'$, the transition from q to q' through the navigation link l . Navigation links can be composed into sequences to form complex navigation links:*

$$q_0 [l_1; \dots; l_n] q_n = q_0 [l_1] q_1 \dots q_{n-1} [l_n] q_n.$$

Definition 9 (navigation consistency) *A navigation link l is consistent w.r.t. a dataset and an initial query q that has answers, i.e., $ext(q) \neq \emptyset$, if the resulting query $l(q)$ also has answers, i.e., $ext(l(q)) \neq \emptyset$. A set of navigation links is consistent iff every navigation link it contains is consistent.*

We now define the three primitive kinds of navigation links and prove under which conditions they are consistent. The zoom-in conjunctively extends the initial query with a complex class.

Definition 10 (zoom-in) *Let q be the initial query, and x be a complex class. The zoom-in on x is a navigation link that leads to the query q and x :*

$$q [\text{zoom-in } x] q \text{ and } x.$$

A zoom-in is consistent iff it is applied to an index term, so that every index term represents a consistent navigation link. Every index term leads to its associated refined query (see Definition 6).

Lemma 11 *Let q be a query, and $index(q)$ its associated index. For every complex class x , the navigation link `[zoom-in x]` is consistent iff $x \in index(q)$.*

Proof. By definition of $index(q)$ and navigation consistency. □

Zoom-in is mostly useful when the extension of the resulting query is strictly smaller. This is obtained when using index terms whose precision is strictly less than 1. This useful distinction can be made visible in the interface by different renderings (e.g., font-color), and annotations (e.g., precision, count).

Naming works similarly to zoom-in, but is expected to apply to a fresh variable, i.e., not occurring in the initial query, while zoom-in is expected to apply to variables already occurring in the initial query.

Definition 12 (naming) Let q be the initial query, and $?v$ be a variable. The naming on $?v$ is a navigation link that leads to the query q and $?v$:

q [naming $?v$] q and $?v$.

Lemma 13 Let q be a query. For every variable $?v$, the navigation link [naming $?v$] is consistent if $?v$ does not occur in the query q .

Proof. If $?v$ does not occur in q , then q and $?v$ is equivalent to q and $?$, which is equivalent to q . Therefore, by the hypothesis that $ext(q) \neq \emptyset$, the navigation link [naming $?v$] is consistent. \square

Naming does not change the extension, because it produces a query that is equivalent to the initial query, but it introduces a new variable in the query, and hence in the index. Subsequent zoom-in navigation links on these variables allow to form cycles in the graph pattern of the query.

Like naming, reversal does not change the graph pattern, but it changes the variable in the SELECT clause. Indeed, in a complex class, the focus is only on one variable, and it is useful to change this focus. A difficulty is that not all variables in the SPARQL pattern appear in its corresponding complex class. In a reversal navigation link, the new variable is implicitly designated by a syntactic element of the query, more precisely by a leaf of its syntax tree.

Definition 14 (reversal) Let q be an initial query, and e be a syntactic element of this query. The reversal on e is a navigation link that leads to the query $\rho(? , q)$, where ρ is defined by induction on complex classes (the underlined part indicates in which part of the query the selected element stands): for every complex property P , and complex classes C, C_1, C_2 ,

- $\rho(q', \underline{?}) = q'$,
- $\rho(q', P \underline{C}) = \rho(P^{-1} q', C)$,
- $\rho(q', \underline{C_1}$ and $C_2) = \rho(q'$ and $C_2, C_1)$,
- $\rho(q', C_1$ and $\underline{C_2}) = \rho(q'$ and $C_1, C_2)$,
- $\rho(q', \underline{C}) = q'$ and C , otherwise.

This definition needs an inverse operation that is defined by induction on complex properties:

$$\begin{array}{lll} (p \text{ :})^{-1} = p \text{ of} & (p \text{ with})^{-1} = p \text{ with} & (\text{opt } P)^{-1} = \text{opt } P^{-1} \\ (p \text{ of})^{-1} = p \text{ :} & & (\text{trans } P)^{-1} = \text{trans } P^{-1} \end{array}$$

We illustrate reversal with two examples, starting with the query already presented earlier: $q = \text{a woman and parent : birth : (date : 1642 and place : in VA)}$.

- q [reversal birth] parent of a woman and birth : (date : 1642 and place : in VA)
result: “a parent of a woman, born in 1642 in Virginia”
- q [reversal in] place of (birth of parent of a woman and date : 1642) and in VA
result: “where in Virginia a parent of a woman was born in 1642”

Lemma 15 A reversal navigation link is necessarily consistent.

Proof. It can be proved that the initial and resulting query of a reversal define the same SPARQL query, up to the renaming of variables, and the possible replacement of the variable in the SELECT clause. If the initial query q has answers, this implies that every variable in the graph pattern has substitution values. Indeed, the graph patterns generated from complex classes contain neither optional patterns, nor union patterns. Therefore, the resulting query q' , which has an equivalent graph pattern, also has answers. \square

We can now define a consistent set of navigation links from a local view.

Definition 16 (local links) Let q be a query defining a local view. The set of local links, noted $links(q)$, is defined as the union of a zoom-in link for each index term $x \in index(q)$, a naming link for one fresh variable, and a reversal link for each syntactic element in the query q .

Theorem 17 (navigation consistency) For every query q such that $ext(q) \neq \emptyset$, the set of local links $links(q)$ is consistent.

Proof. This is a direct consequence of previous definitions and lemmas. \square

This implies that, given the initial query has answers, a user that only follows local links will never fall in a dead-end, which is a frequent cause of frustration in information systems. An important related issue is completeness, i.e., the ability to reach arbitrary queries by navigation only, rather than with the help of manual query editing.

Definition 18 (navigation completeness) *A definition of local links is complete iff for every query q whose extension is not empty, there exists a finite sequence of navigation links $q_0 [l_1] q_1 \dots q_{n-1} [l_n] q_n$, where $q_0 = ?$, $q_n = q$, and for every $i \in [1, n]$, the navigation link l_i is a local link of its initial query: $l_i \in \text{links}(q_{i-1})$. We call such a sequence, a sequence of local links.*

If we have an unlimited index, then navigation is trivially complete. Indeed, if a query q verifies $\text{ext}(q) \neq \emptyset$, then it belongs to the index of the query $?$ because q is equivalent to $?$ and q . Therefore, q can be reached in one zoom-in link. What is interesting is to prove navigation completeness for a finite index.

Theorem 19 (finite complete index) *Let $\text{base_index}(q)$ be the subset of $\text{index}(q)$ restricted to resources (IRIs, literals), variables occurring in q , and the restrictions $P ?$. The set of local links derived from this base index is finite and complete for navigation.*

Proof. The base index is finite because in a given dataset (1) there is a finite number of IRIs, and hence of properties, (2) only literals present in the extension belong to the index, and this extension is always finite, and (3) every query has a finite number of variables.

For completeness, we have to prove that every zoom-in on an index term that is not in the basic index can be decomposed into a sequence of local links that are derived from the base index. This is achieved by an inductive reduction of the excluded index terms (\equiv denotes query equivalence):

- q [zoom-in $?v$] q and $?v = q$ [naming $?v$] q and $?v$, for every variable $v \notin q$
- q [zoom-in $?$] q and $? = q$ [] ($q \equiv q$ and $?$) ([] is the empty sequence)
- q [zoom-in $P C$] q and $P C = q$ [zoom-in $P ?$] q and $P ?$ [reversal $?$] ($? \text{ and } P^{-1} q \equiv P^{-1} q$)
[zoom-in C] $P^{-1} q$ and C [reversal q] q and $P C$
[zoom-in $P ?$] is a local link (from the base index) because $P ?$ is more general than $P C$, which is a local link by hypothesis. [zoom-in C] is also a local link, but not necessarily from the base index (needs to be decomposed further).
- q [zoom-in $a i$] q and $a i = q$ [zoom-in $\text{rdf:type} : i$] (q and $\text{rdf:type} : i \equiv q$ and $a i$)
- q [zoom-in C_1 and C_2] q and $(C_1 \text{ and } C_2) = q$ [zoom-in C_1] q and C_1 [zoom-in C_2] ($(q \text{ and } C_1) \text{ and } C_2 \equiv q \text{ and } (C_1 \text{ and } C_2)$)
If q and $(C_1 \text{ and } C_2)$ has answers, then q and C_1 has also answers, because it is more general. Therefore, $C_1 \in \text{index}(q)$, and $C_2 \in \text{index}(q \text{ and } C_1)$. \square

In the base index, the set of resources can be seen as the list of answers to the query, and can be presented page by page, like in web search engines. The set of properties is organised into a subsumption hierarchy that can be expanded on demand by users. Instead of showing up to 12 complex properties for every basic property p , only $p :$ and $p \text{ of}$ can be displayed. Their various closures are accessible by toggling each closure on/off when applying zoom-in (see check-boxes in Figure 2).

The index computed in our prototype is richer than the base index, in order to provide richer summaries of query results. It contains the hierarchy of classes ($a i$), and the user can expand restrictions recursively, i.e., each $P ?$ is refined into $P C$, where C is derived in the same way the main index is derived from $?$. Zoom-in is performed by double-clicking an index term, naming is performed by entering a variable name in a text field, and reversal is performed by clicking on a query element. The interface offers three short-hand navigation links on index terms (see buttons in Figure 2):

- home: reset the query to $?$,
- pivot $C = \text{home}$; zoom-in C ,
- cross $P C = \text{zoom-in } P C$; reversal C .

Then, the above query `a woman and parent : birth : (date : 1642 and place : in VA)` is accessible from ? through the following sequence of navigation links: `zoom-in a woman`; `cross parent : ?`; `cross birth : ?`; `zoom-in date : 1642`; `zoom-in place : in VA`; `reversal a woman`. After selecting women, the user moves to their parents, and then to the birth of their parents. By expanding recursively index terms, the user discovers birth dates and places, and select a date (1642), and a place (in VA). Finally, the user comes back to the point of view of women, now restricted to those whose some parent was born in 1642 in Virginia.

4 Discussion and Conclusion

The contribution of this paper is not about a new query language, but about an interactive navigation process that guides users from query to query, along the LIS principles. The query language proposed here has been driven by LIS constraints (a query denotes a complex class), and the wish to have queries as concise and natural as possible. However, it is interesting to discuss further its expressivity compared to SPARQL [PAG06]. The missing graph patterns are the OPTIONAL, UNION, and FILTER patterns. In our case, the OPTIONAL pattern is useless because the SELECT clause has only one variable. Our prototype has already UNION patterns through an `or` operator but we do not have consistency and completeness results yet about their navigation. It also has negation, and some limited forms of FILTER patterns as predefined classes of literals. For instance, the class `match "regexp"` denotes the set of all strings that match a regular expression. Similarly, we have classes for intervals and inequalities over numbers and dates. The most important restriction in our queries is the one-variable SELECT clause. However, the index alleviates this restriction to some extent. Suppose the SPARQL query `SELECT ?x ?y WHERE { ?x rdf:type gen:man . ?x gen:mother ?y }`. By setting the query to `a man`, and by expanding `mother : ?`, the index gives for each mother, how many male children she has. A highlighting mechanism allows to select a man in the extension to discover who is his mother; and alternately, to select a mother in the index to discover which are her children. The index is an inverted view over the table of SPARQL results. Each subtree of the index (with *count* annotation) is a histogram of the values from a column of the table. The highlighting mechanism enables to retrieve the associations between the values of the different columns, i.e., the result tuples.

SPARQL defines updates and rules by using the same graph patterns as in queries. In the same way, we plan to reuse complex classes for specifying updates and rules. Like for queries, we do not expect from users to type updates and rules, and we aim to reuse the local view and navigation links to guide users in the expression of updates and rules.

We have defined local views over RDF(S) datasets that serve both for summarization and navigation. The set of navigation links available in local views is both consistent and complete, thus combining the expressivity of querying languages with the ease-of-use of navigation. Important remaining issues are scalability and ergonomy. Our current prototype is an evolution of an existing LIS (CAMELIS), and has not yet been optimized w.r.t. these issues. About scalability, everything in our approach can be reduced to answering queries, so that existing results about efficiently answering queries can be reused. If one wants to include more reasoning in the process, it is possible to translate complex classes and properties to OWL query languages (e.g., SPARQL-DL [SP07], OWL-QL [FHH04]), and reuse existing reasoners. About ergonomy, the index is a new and rich structure that requires and suggests new interfaces. For instance, index terms about dates may be rendered on a calendar, showing graphically the distribution of events over time. The same can be done with places on a map. Like the index, these graphical representations would be activable. For instance, the selection of an area on the map would trigger a zoom-in navigation link [BFRQ08].

References

- [AF08] P. Allard and S. Ferré. Dynamic taxonomies for the semantic web. In G.M. Sacco, editor, *Int. Work. Dynamic Taxonomies and Faceted Search (FIND)*, pages 382–386. IEEE Computer Society, 2008.
- [BCM⁺03] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
- [BFRQ08] O. Bedel, S. Ferré, O. Ridoux, and E. Quesseveur. GEOLIS: A logical information system for geographical data. *Revue Internationale de Géomatique*, 17(3-4):371–390, 2008.
- [CKZ04] O. Corby, R. Dieng Kuntz, and C. Faron Zucker. Querying the semantic web with corese search engine. In R. López de Mántaras and L. Saitta, editors, *Eu. Conf. Artificial Intelligence*, pages 705–709. IOS Press, 2004.
- [Fer09] S. Ferré. Camelis: a logical information system to organize and browse a collection of documents. *Int. J. General Systems*, 38(4), 2009.
- [FHH04] R. Fikes, P. J. Hayes, and I. Horrocks. OWL-QL - a language for deductive query answering on the semantic web. *J. Web Semantic*, 2(1):19–29, 2004.

- [FR04] S. Ferré and O. Ridoux. An introduction to logical information systems. *Information Processing & Management*, 40(3):383–419, 2004.
- [GW99] B. Ganter and R. Wille. *Formal Concept Analysis — Mathematical Foundations*. Springer, 1999.
- [Mar06] G. Marchionini. Exploratory search: from finding to understanding. *Communications of the ACM*, 49(4):41–46, 2006.
- [PAG06] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of sparql. In I. F. Cruz *et al*, editor, *Int. Semantic Web Conf.*, pages 30–43, 2006.
- [PAG08] J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF. In A. P. Sheth *et al*, editor, *Int. Semantic Web Conf.*, pages 66–81, 2008.
- [PR03] Y. Padiou and O. Ridoux. A logic file system. In *Usenix Annual Technical Conference*, pages 99–112. USENIX, 2003.
- [SP07] E. Sirin and B. Parsia. SPARQL-DL: SPARQL query for OWL-DL. In C. Golbreich, A. Kalyanpur, and B. Parsia, editors, *Work. OWL Experiences and Directions (OWLED)*, volume 258. CEUR-WS, 2007.
- [ST09] G. M. Sacco and Y. Tzitzikas, editors. *Dynamic taxonomies and faceted search*. The information retrieval series. Springer, 2009.
- [vR86] C. J. van Rijsbergen. A new theoretical framework for information retrieval. In *Int. ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 194–200. ACM, 1986.