

Dynamic Taxonomies for the Semantic Web

Pierre Allard and Sébastien Ferré

IRISA/Université de Rennes 1
Campus de Beaulieu
35042 Rennes cedex, France
piallard@irisa.fr, ferre@irisa.fr

Abstract

The semantic web aims at enabling the web to understand and answer the requests from people and machines. It relies on several standards for representing and reasoning about web contents. Among them, the Web Ontology Language (OWL) is used to define ontologies, i.e. knowledge bases, and is formalized with description logics. In this paper, we demonstrate how dynamic taxonomies and their benefits can be transposed to browse OWL-DL ontologies. We only assume the ontology has an assertional part, i.e. defines objects and not only concepts. The existence of relations between objects in OWL leads us to define new navigation modes for crossing these relations. A prototype, ODALISQUE, has been developed on top of well-known tools for the semantic web.

1. Introduction

Dynamic taxonomies (DT) have proven their usefulness to browse medium-to-large information bases [9]. Objects can be classified under an arbitrary number of concepts; and concepts are organized in a multi-dimensional taxonomy. This taxonomy supports both expressive querying and flexible navigation. A query is any boolean combination of concepts, and defines a focus over the information base. Every query determines an *extension* as the set of answers to the query, and a *dynamic taxonomy* as the pruning of the taxonomy to those concepts that share objects with that extension. The dynamic taxonomy serves both as a summary of the current focus, and as a set of navigation links to reach other foci. A navigation link combines the current query and a selected concept to form a new query, and hence reach a new focus. DTs enable people with no *a priori* knowledge of an information base to acquire such a knowledge, and to reach relevant objects, as demonstrated by their use in e-commerce applications [10].

An interesting question is whether DTs can be adapted to more complex information bases than object collections and taxonomies. A step in this direction is made by Logical Information Systems (LIS) that use almost arbitrary logics instead of taxonomies [2, 3]. Values and patterns can be used in addition to concept names in object descriptions, queries and dynamic taxonomies; and their organization into a taxonomy is automatically derived by logical inference. They can also be combined to form more complex concepts such as coordinates, or tree patterns. However, a persisting limitation is that the information base is essentially a collection of objects that are unrelated to each other, apart from sharing common concepts. The consequence is a biased representation in many applications. For instance, in a bibliographic application, objects can be either documents or authors, but not both. If objects are documents, then authors must be concepts, and hence cannot be classified themselves (e.g., institution, discipline). If objects are authors, then documents must be concepts, and hence cannot be classified themselves (e.g., publication year and venue). If both documents and authors were objects, then there would be no way in DTs to relate them.

Description Logics (DL) are logical formalisms that have been adopted to represent and reason about ontologies [6], which are domain-specific knowledge bases. They have points in common with DTs such as concepts, subsumption between concepts (similar to taxonomic relation), and instance relation between objects and concepts. There are however several important differences. First, concepts are not only concept names, but complex combinations of concepts by logical connectors. Second, the subsumption relations are derived automatically from a set of axioms modelling the domain. Third, relations (called *roles*) can be set between objects, so that the concepts to which an object belongs depend on other objects to which it is connected. The two first differences are shared by LIS to some extent, but the third is not.

In this paper, we show how DTs, and LIS, can be ex-

tended so as to cope with semantic web ontologies, while retaining all their good properties. In particular, the presence of roles between objects leads us to define new navigation modes. In Section 2 we shortly introduce web ontologies and description logics. In Section 3 we redefine the notions of query, extension and dynamic taxonomies that make up a focus. In Section 4 we redefine the navigation links for zoom-in, zoom-out and pivot, and we introduce navigation links for crossing roles: *reversal* and *traversal*. Section 5 presents our prototype ODALISQUE, and illustrates it over an example ontology. Section 6 concludes and draws perspectives.

2. Web Ontologies

The amount of data in the web is growing day by day, giving access to more and more information. The main problem is looking for a specific information in this flow. Indeed, information is generally not given a formal semantics, and hence is not understandable to a computer. This is the main reason for the semantic web. Initial works fixed the formal syntax, XML. Its main problems are its lack of semantics, and its lack of relations between entities. That is why other standards have been developed: RDF (Resource Description Framework), then the more expressive DAML+OIL (fusion of DAML-ONT and OIL). This last language is the starting point of OWL [5], the Web Ontology Language, the W3C standard¹.

The purpose of OWL is to define ontologies for modelling and reasoning about application domains. OWL is mapped onto the description logics. Description logics provides the required theoretical background, such as proving the consistency of an ontology, or classifying concepts. The OWL standard includes 3 expressivity levels: OWL Lite, OWL DL and OWL Full. For example, OWL Lite does not contain the notion of cardinality on roles, but inference is decidable and efficient. OWL Full uses all the primitives of OWL, then it is the most expressive (and fully includes RDF), but becomes undecidable. In this paper, we are interested in OWL DL, because it is decidable, a good compromise between expressivity and inference efficiency, and well supported by most existing tools (unlike OWL Full). In the following, we recall the basic definitions of OWL DL, only to the extent of what is required in this paper. Here, we adopt the terms of descriptions logics rather than those of the OWL standard, because they provide a much more compact syntax, and because our focus is more on theory than on technology.

¹<http://www.w3.org/2004/OWL/>

2.1 Description Logics

The definition of description logics is based on three main ideas: concepts, roles and objects. Given a domain of individuals (e.g., persons in genealogy, documents in bibliography), the concepts are sets of individuals having shared characteristics (e.g. *Women*); the roles are binary relations between individuals (e.g. *sibling*); the objects are particular individuals (e.g. BOB). The syntax gathers these 3 main ideas, plus constructors for building complex concepts and roles, which determine the expressivity of the description logic.

Definition 1 (Signature) *The language of formulas of a description logic is characterized by a signature $S = (O, C_a, R_a, Cstr)$, where:*

- O is a set of objects;
- C_a is a set of atomic concepts (denoted c_i);
- R_a is a set of atomic roles (denoted r_i);
- $Cstr$ is a set of constructors used to create complex concepts (denoted C_i) and complex relations (denoted R_i). Classes of constructors of the common description logics are represented by letters.

The description logic OWL DL is based on the description logic \mathcal{SHOIQ} , whose constructors are listed in Table 1 (the notation $\#S$ represents the cardinality of a set S). This list of constructors determines the expressivity of OWL DL.

Once the syntax is established, the second notion to define is semantics. The main difference between classical logic and description logics is the interpretation function. Where classical logic returns for each formula a truth value, description logics interpretation function returns for each concept a set of individuals, its instances, and for each role a binary relation between individuals. The interpretation function for the constructors of the description logic \mathcal{SHOIQ} is given in Table 1.

Definition 2 (Interpretation) *Let $S = (O, C_a, R_a, Cstr)$ be a signature. An interpretation $\mathcal{I} = (\Delta_{\mathcal{I}}, \cdot^{\mathcal{I}})$ of S is a set of individuals $\Delta_{\mathcal{I}}$, called the interpretation domain, and a interpretation function $\cdot^{\mathcal{I}}$, that associates:*

- to each object o_i an individual $o_i^{\mathcal{I}} \in \Delta_{\mathcal{I}}$;
- to each concept c_i a set of individuals $c_i^{\mathcal{I}} \subseteq \Delta_{\mathcal{I}}$;
- to each relation r_i a binary relation between individuals $r_i^{\mathcal{I}} \subseteq \Delta_{\mathcal{I}} \times \Delta_{\mathcal{I}}$.

Letter	Syntax	Definition	Interpretation
S	top (most general concept)	\top	$\Delta_{\mathcal{I}}$
	bottom (most specific concept)	\perp	\emptyset
	conjunction of two concepts	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
	disjunction of two concepts	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
	concept negation	$\neg C$	$\Delta_{\mathcal{I}} \setminus C^{\mathcal{I}}$
	qualified universal quantification	$\forall r.C$	$\{i \in \Delta_{\mathcal{I}} \mid \forall j : (i, j) \in r^{\mathcal{I}} \rightarrow j \in C^{\mathcal{I}}\}$
	qualified existential quantification	$\exists r.C$	$\{i \in \Delta_{\mathcal{I}} \mid \exists j : (i, j) \in r^{\mathcal{I}} \wedge j \in C^{\mathcal{I}}\}$
O	at least one of objects	$\{o_1, \dots, o_n\}$	$\{o_1^{\mathcal{I}}, \dots, o_n^{\mathcal{I}}\}$
Q	minimal cardinality	$\geq n r.C$	$\{i \in \Delta_{\mathcal{I}} \mid \#\{j \in C^{\mathcal{I}} \mid (i, j) \in r^{\mathcal{I}}\} \geq n\}$
	maximal cardinality	$\leq n r.C$	$\{i \in \Delta_{\mathcal{I}} \mid \#\{j \in C^{\mathcal{I}} \mid (i, j) \in r^{\mathcal{I}}\} \leq n\}$
	exact cardinality	$= n r.C$	$\{i \in \Delta_{\mathcal{I}} \mid \#\{j \in C^{\mathcal{I}} \mid (i, j) \in r^{\mathcal{I}}\} = n\}$

Table 1. Set of constructors of the description logic \mathcal{SHOIQ} , with their syntax and interpretation.

The third notion to be defined for a description logic is the knowledge base. A knowledge base divides the knowledge in two parts: the *terminological* part, which represents general knowledge, true for all individuals, and the *assertional* part, which represents particular knowledge, only true for particular individuals.

Definition 3 (Knowledge base) We define a knowledge base as a pair $\Sigma = (T, A)$, where:

- T is the terminological part (T-Box), represented by a set of axioms like “ C_i is subsumed by C_j , denoted $C_i \sqsubseteq C_j$, which means that every instance of C_i is necessarily an instance of C_j . The equivalence of 2 concepts, denoted by $C_i \doteq C_j$, is defined by the 2 axioms $C_i \sqsubseteq C_j$ and $C_j \sqsubseteq C_i$;
- A is the assertional part (A-Box), represented by a set of assertions like $o : C$ and $(o_i, o_j) : R$, which respectively means that o belongs to the concept C , and a connection R exists between o_i and o_j .

Moreover, the description logic \mathcal{SHOIQ} allows additional axioms in the knowledge base. The following axioms on roles can be added to the T-Box:

- $r_i \sqsubseteq r_j$ (called role hierarchy);
- r_i inverse of r_j ;
- r functional;
- r transitive.

In the following, the notation r^{-1} is used to designate the role that is defined to be the inverse of the role r , when defined.

The Figure 1 presents the T-Box of an example knowledge base Σ_{ex} , with 3 concepts: *Person*, *Team*, and *Bigteam*. A instance of *Team* is defined as having a leader and at least 2 members. An instance of *Bigteam* is defined

$Person \sqsubseteq$	\top
$Team \doteq$	$\exists \text{hasleader}.Person$ $\sqcap \geq 2 \text{hasmember}.Person$
$Bigteam \doteq$	$\exists \text{hasleader}.Person$ $\sqcap \geq 3 \text{hasmember}.Person$
$\text{hasleader} \sqsubseteq$	hasmember

Figure 1. Example of T-Box

OLIVIER :	<i>Person</i>
SEBASTIEN :	<i>Person</i>
PIERRE :	<i>Person</i>
LIS :	<i>Team</i>
(LIS, OLIVIER) :	<i>hasleader</i>
(LIS, SEBASTIEN) :	<i>hasmember</i>
(LIS, PIERRE) :	<i>hasmember</i>

Figure 2. Example of A-Box

as having a leader and at least 3 members. To help the reading of knowledge bases, concepts are written with an uppercase, relations in lowercase, and objects in uppercase letters. Figure 2 presents the A-Box of Σ_{ex} , with the description of the team LIS, led by OLIVIER and composed of 2 others members, PIERRE and SEBASTIEN.

Given a knowledge base Σ , some interpretations are distinguished as *models* of Σ .

Definition 4 (Model) Let $\Sigma = (T, A)$ be a knowledge base and \mathcal{I} an interpretation, for the same signature S . We call \mathcal{I} model of Σ , denoted $\mathcal{I} \models \Sigma$, when:

- for each $C_i \sqsubseteq C_j$ in T , $C_i^{\mathcal{I}} \subseteq C_j^{\mathcal{I}}$;
- for each $o : C$ in A , $o^{\mathcal{I}} \in C^{\mathcal{I}}$;
- for each $(o_i, o_j) : R$ in A , $(o_i^{\mathcal{I}}, o_j^{\mathcal{I}}) \in R^{\mathcal{I}}$.

The definition of *SHOIQ* gives additional properties to be checked:

- for each $r_i \sqsubseteq r_j$ in T , $r_i^{\mathcal{I}} \subseteq r_j^{\mathcal{I}}$;
- for each r_i inverse of r_j in T ,
 $r_i^{\mathcal{I}} = \{(i, j) \in \Delta_{\mathcal{I}} \times \Delta_{\mathcal{I}} \mid (j, i) \in r_j^{\mathcal{I}}\}$;
- for each r functional in T ,
 $(i, j) \in r^{\mathcal{I}} \wedge (i, j') \in r^{\mathcal{I}} \rightarrow j = j'$;
- for each r transitive in T ,
 $(i, j) \in r^{\mathcal{I}} \wedge (j, k) \in r^{\mathcal{I}} \rightarrow (i, k) \in r^{\mathcal{I}}$.

From this notion of model, statements and their inference from a knowledge base can be defined: classification and instantiation. The *classification* statements establish subsumption relations between concepts, and hence help to organize or classify them. The *instantiation* statements establish the belonging of objects to concepts, and hence help to place objects in the concept classification.

Definition 5 (Classification and Instantiation) Let Σ be a knowledge base. Two kinds of statements can be inferred from Σ :

- *Classification*: $\Sigma \models C_i \sqsubseteq C_j$ iff $\mathcal{I} \models C_i \sqsubseteq C_j$, for every model \mathcal{I} of Σ ;
- *Instantiation*: $\Sigma \models o : C$ iff $\mathcal{I} \models o : C$, for every model \mathcal{I} of Σ .

For instance, it is easy to prove in our example that every big team is also a team: $\Sigma_{ex} \models \text{Bigteam} \sqsubseteq \text{Team}$. Also, OLIVIER being the leader of LIS, hence a member of it, and SEBASTIEN and PIERRE being members too, we can prove that LIS is a big team, i.e. $\Sigma_{ex} \models \text{LIS} : \text{Bigteam}$.

2.2 Browsing Ontologies

There exists a number of tools for browsing ontologies. Some of them are simply viewers that make no or little inference. They present an ontology in the form of a tree or a graph. The tree represents the classification of all named concepts, and each concept can also be decorated by its instances (e.g., Protégé). The graph represents a knowledge base by representing concepts and objects by nodes, and subsumption, equivalence, and assertions by edges (e.g., RDF-Graph-Viewer). On one hand, some viewers impose to display the full ontology at once, making it quickly unreadable. On the other hand, some viewers display only predefined subsets of information, e.g. the A-Box or the T-Box (e.g., SVG-OWL-Viewer). In summary, these viewers do not provide navigation facilities that would allow users to focus on various subsets of information.

Other tools provide querying facilities, by adapting the SQL language to web ontologies. OntoQL [7] permits to query Ontology-Based Databases (OBDB); RQL [8] is designed to query RDF ontologies; OWL-QL [4] is a language to query OWL ontologies. They heavily rely on inference mechanisms in the course of computing answers to queries. However, like with most querying systems, the user is not guided in his search, and must have preliminary knowledge about the ontology structure, and the query language, in order to express relevant queries.

3. Local View: Query, Extension, and Dynamic Taxonomy

The objective of this section is to show how a web ontology, i.e. a knowledge base as defined in Section 2, can fit into the framework of dynamic taxonomies. The key notions are the classification and instantiation statements that can be inferred from a knowledge base. Classification statements define a subsumption ordering over concepts, which can play the role of a taxonomy. Indeed, this is a multi-dimensional taxonomy because an individual can belong to an arbitrary number of concepts, and concepts can have several super-concepts. Instantiation statements determine whether an object is an instance of a concept, and hence define the deep extension of concepts. Therefore, all ingredients are present to apply the framework of dynamic taxonomies on web ontologies. The important differences with the usual application of DTs is that (1) the concepts may be complex and in infinite number, and (2) the taxonomy shape and extensions are defined through logical inference. The infinite number of concepts is coped with by selecting which concepts should appear in taxonomies, and also by computation-on-demand when expanding concepts. Roles seem absent from this picture, but in fact, they appear as parts of complex concepts and are exploited by new kinds of navigation links (see Section 4).

At each focus, we define the *local view* as the combination of a query, an extension, and a dynamic taxonomy. The *query* specifies the current focus, i.e. the user location in the vast navigation space, or her point of view over the whole knowledge base. The *extension* is the set of answers of the query, i.e. a set of objects. The *dynamic taxonomy* gives feedback about this extension, and it is the support of most navigation links. These 3 components of local views are formally defined in the following, on top of description logics.

Definition 6 (Query) Let S be a signature. A query is a (complex) concept, which can be written with all OWL DL constructors. In particular, the 3 boolean constructors are available (conjunction, disjunction, negation).

Sometimes, it is useful to see the query as a conjunctive set of simpler concepts, which can be obtained from any query by putting it in conjunctive normal form. In this paper, we use alternately the 2 forms, whichever is the most convenient. For instance, the query $q = Team \sqcap \geq 6 \text{ hasmember. Person}$ is identical to $q = \{Team, \geq 6 \text{ hasmember. Person}\}$.

Next, given a query q , the extension is defined as the set of objects that can be proved to be instances of q .

Definition 7 (Extension) *Let Σ be a knowledge base and q be a query. The extension of q is the set of objects which are instances of this concept:*

$$\text{ext}(q) = \{o \in O \mid \Sigma \models o : q\}.$$

This definition explains why we assume the knowledge base has an A-Box (assertional box). Otherwise, no instantiation statement could be inferred, and so all extents would be empty. An important property of extensions is that they are monotonic w.r.t. subsumption. For every concepts C, D , if C is subsumed by D , then the extension of C is included in the extension of D :

$$\Sigma \models C \sqsubseteq D \Rightarrow \text{ext}(C) \subseteq \text{ext}(D).$$

This is consistent with dynamic taxonomies, where we observe the same relation between taxonomic relations and (deep) extensions.

Before defining the dynamic taxonomy associated to a query, we first have to define the taxonomy itself. It is made of a subset of the concept language, ordered by subsumption.

Definition 8 (Taxonomy) *Let Σ be a knowledge base. The taxonomy derived from Σ is the partially ordered set $T_\Sigma = (X_\Sigma, \sqsubseteq_\Sigma)$. The set of concepts X_Σ is the smallest set that contains:*

- the concept \top ;
- all concept names in C_a ;
- for every role name $r \in R_a$, every concept name $c \in C_a$, and every natural number $n \in \mathbb{N}$, the concepts $\exists r. \top$, $\exists r. c$, $\geq n r. \top$ and $\geq n r. c$.

Any 2 concepts $C, D \in X_\Sigma$ are ordered by \sqsubseteq_Σ , i.e. $C \sqsubseteq_\Sigma D$ iff $\Sigma \models C \sqsubseteq D$.

The 3 boolean constructors (\sqcap , \sqcup , and \neg) are excluded from the taxonomy because they are easily introduced into queries through the navigation process (see Section 4). The “at least one of objects” can also be introduced through navigation by a direct selection of objects in the extension. Other constructors are excluded from the taxonomy because they are redundant according to the following equivalences:

- $\forall r. C \doteq \neg \exists r. \neg C$;
- $\leq n r. C \doteq \neg \geq (n+1) r. C$;
- $= n r. C \doteq \geq n r. C \sqcap \leq n r. C$.

Another motivation for not having the constructors \forall , \leq and $=$ is the Open World Assumption (OWA). Indeed, description logics, hence OWL, work under this assumption. For example, in our knowledge base Σ_{ex} , we have declared 3 objects as persons and members of the team LIS. The OWA implies that there could be other members of LIS, unknown to the knowledge base designer. Hence, the team LIS is not an instance of the concept $\leq 5 \text{ hasmember. Person}$ (“at most 5 members”), whose extension is empty in our knowledge base. In fact, this extension is empty in most practical knowledge bases, except if a team is explicitly declared to have at most 5 members or less. By excluding these rare concepts from the taxonomy, we make it more compact and efficient, while retaining the ability to insert them manually in queries.

Now, given a query q , this taxonomy is pruned to retain only concepts that are extensionally related to the query, which results in the *dynamic taxonomy*.

Definition 9 (Dynamic taxonomy) *Let Σ be a knowledge base, and q be a query. A concept $x \in T_\Sigma$ is extensionally related to the query q iff*

$$\text{ext}(x) \cap \text{ext}(q) \neq \emptyset.$$

This definition can be refined by using a minimal support m (putting $m = 1$ is equivalent to the last definition):

$$\#(\text{ext}(x) \cap \text{ext}(q)) \geq m.$$

The dynamic taxonomy $DT_\Sigma(q)$ is the pruning of the taxonomy T_Σ to the concepts that are extensionally related to the query q . For historical reasons [2], we name those concepts increments of q .

Because of concepts $\geq n r. C$, there is an infinite number of concepts in the taxonomy T_Σ . However, every dynamic taxonomy is finite because a knowledge base is finite, and therefore for every role r and every concept C , there is necessarily a number k such that for every $n \geq k$, the extension of $\geq n r. C$ is empty. Still, dynamic taxonomies are often too large to be computed and displayed entirely at once. This is why users are initially presented with a fully collapsed dynamic taxonomy, showing only the most general concept \top , and are allowed to expand concepts on demand, i.e. computing and displaying children increments. This computation is performed by the function $\text{incrs}(x, q, m)$ that returns the children of the concept x in $DT_\Sigma(q)$, given the current query q and a minimum support m .

We now sketch the definition of the function $\text{incrs}(x, q, m)$, depending on the shape of the parent concept x :

- $x \doteq \top$: x is the root of the dynamic taxonomy. We must open the main ways, that is to say, return:
 - the most general concept names in C_a ,
 - for every most general role r in R_a (and their inverses), the concept $\exists r.\top$;
- $x \in C_a$: x is a concept name. We must return the most general concept names in C_a that are subsumed by x ;
- $x = \geq n r.C$: x is a qualified cardinality (this includes every concept $\exists r.C$ as equivalent to $\geq 1 r.C$). We must return:
 - for every most general concept name $D \in C_a$ subsumed by C , the concept $\geq n r.D$,
 - for every most general role name $s \in R_a$ subsumed by r , the concept $\geq n s.C$,
 - the concept $\geq (n+1) r.C$.

In fact, among returned concepts, only those that are extensionally related to the query (i.e. increments) are retained, which requires the computation of an extension and an intersection for each concept.

More pruning can be done with the help of the T-Box. For instance, knowing r is a functional role, it is not necessary to return the increment $\geq 2 r.\top$, because its extension will necessarily be empty. Other prunings can be done with *inverse functional* or *transitive* roles. Also, concepts in the form $\exists r.\top$ could be made more informative by replacing \top by the concept describing the range of the role r (e.g. $\exists hasmember.Person$ instead of $\exists hasmember.\top$).

4. Navigation Links

A focus is a particular point of view over a knowledge base, and we need navigation links to move from one focus to another. We distinguish 3 types of navigation links: zoom-in/out, pivot, and crossing roles. Every navigation link is anchored on some element of the local view, and defined as a function that maps the current query to a new query. All these navigation links are defined in the following, and are illustrated in a navigation scenario (see Section 5.3).

4.1 Zoom-in and Zoom-out

The most common navigation link is the zoom-in link. A zoom-in link, aka specialization or refinement, reaches a query, whose extension is smaller than the previous extension, but not empty. A concept from the dynamic taxonomy $DT_\Sigma(q)$, an increment of q , can be used as a zoom-in link iff $ext(x) \cap ext(q) \subsetneq ext(q)$, in order to ensure that a

strictly smaller extension is reached. Let x be such an increment, the new query is obtained by replacing by x the elements in q that subsume x (specialization):

$$q \leftarrow (q \setminus \{C \in q \mid x \sqsubseteq C\}) \cup \{x\}.$$

A zoom-out link, aka generalization, is the inverse of the zoom-in. A increment x can be used as a zoom-out link iff it subsumes some query element: $\exists C \in q : C \sqsubseteq x$. There are 2 cases when following a zoom-out link, respectively the removal and the generalization of query elements. In the first case, when $x \in q$, the concept x is simply removed from the query:

$$q \leftarrow q \setminus \{x\}.$$

In the second case, when $x \notin q$, the query elements that are strictly subsumed by x are replaced by x (generalization):

$$q \leftarrow (q \setminus \{C \in q \mid C \sqsubseteq x\}) \cup \{x\}.$$

Hence, every zoom-in link can be undone by a zoom-out link, and conversely.

To increase the expressivity of navigation links, we allow the combination of increments into complex increments. First, when several increments $\{x_i\}_{i \leq n}$ are selected in the dynamic taxonomy, they are disjuncted to form a complex increment $x = x_1 \sqcup \dots \sqcup x_n$. Second, a negated zoom-in is provided by replacing x by $\neg x$ in the above definition.

Finally, the OWL DL logic also provides the constructor $\{o_1, \dots, o_n\}$, whose extension is trivially the set $\{o_1, \dots, o_n\}$. These concepts can be used in zoom-in and zoom-out links exactly like other increments. The difference is that, instead of picking concepts in the dynamic taxonomy, the user has to pick objects in the extension. In this way, the extension can be restricted to a subset of the current extension (zoom-in) or a set of objects can be excluded from the current extension (negated zoom-in).

4.2 Pivot

The navigation link, called *pivot* link, is the most simple and consists in replacing the current query q by some increment x in the dynamic taxonomy $DT_\Sigma(q)$:

$$q \leftarrow x.$$

Pivot links are useful to change from one point of view to another that is extensionally related. Like for zoom-in, disjunction, negation, and object selection can also be used to define the increment.

4.3 Reversal and Traversal

We now introduce and motivate new navigation links that make use of roles. First of all, roles are already

present in dynamic taxonomies through quantified concepts (e.g., $\exists r.C$ and $\geq n r.C$). So, they can be introduced in queries by zoom-in and pivot links. For instance, the following query can be reached in 2 zoom-in steps: $q = \{Bigteam, \exists hasleader.Person\}$, i.e. “the big teams with a leader”. A useful navigation link would be to reach the related query: $\{Person, \exists isleaderof.Bigteam\}$, i.e. “the leaders of big teams”, where *isleaderof* is defined as the inverse of *hasleader*. This navigation link, called *reversal* link, changes the point of view by crossing a role in the query, and turning upside-down the query accordingly. In the above example, the point of view has been changed from teams to persons, through the role *hasleader*.

A query element x can be used as a reversal link if it has the form $x = \exists r.C$, and then the new query is defined as:

$$q \leftarrow \{C\} \cup \{\exists r^{-1}.(q \setminus \{\exists r.C\})\}.$$

It can be verified that, if we follow again the reversal link on $\exists r^{-1}.(q \setminus \{\exists r.C\})$, we come back to the initial query q .

The second type of navigation link using roles is a composition of two navigation links defined above and is called *traversal*. Indeed, it often happens that the user wants to cross a role that is not yet present in the query, but already visible in the dynamic taxonomy. Hence, for an increment $x = \exists r.C$, x is successively used for zoom-in and reversal. These two steps can be simplified by the following definition of the traversal link:

$$q \leftarrow \{C\} \cup \{\exists r^{-1}.q\}.$$

For instance, we can move in one step from the query *Bigteam* to the query $\{Person, \exists isleaderof.Bigteam\}$ by traversing the increment $\exists hasleader.Person$.

In the definitions of reversal and traversal, we assume that the role r has an inverse that is defined in the knowledge base. If a role has no inverse, we can either prevent it to be reversed or traversed, or we can automatically define an inverse role in the knowledge base.

5. Implementation and Examples

We developed a prototype, ODALISQUE, to experiment DTs-like browsing of OWL ontologies. It shares a similar user interface of an existing software, Camelis [1] (a LIS implementation), but is based on a DL reasoner to infer classification and instantiation statements. It also adds the navigation links related to roles. In the following, we demonstrate its use on an ontology about our research laboratory.

5.1 ODALISQUE

The prototype is called ODALISQUE, for Ontology Description and Logical Information System Queries. It uses

the Jena API², developed in JAVA. Jena provides several tools for OWL, especially the parsing of OWL files, and the interaction with a reasoner such as Pellet. This interaction gives access to the two main statements we need: classification and instantiation. ODALISQUE has a GUI, shown in Figure 3, that provides a complete display of local views, and all navigation modes presented in Section 4. At the top, the current query (here $\{Person, \exists isleaderof.Bigteam\}$), can be read and edited. At the left, the dynamic taxonomy is represented as a tree, and a click on a tree node selects it as the current increment. Between the query and the dynamic taxonomy, there is a button for each type of navigation links, and only those that are applicable to the selected increment are activable. At the right, the extension of the current query is displayed as a list. The bold elements are those that would remain if a zoom-in were applied on the selected increment. Several increments can be selected at the same time, which is equivalent to have a single increment defined as the disjunction of those increments. By selecting a set of objects in the extension, the selected increment is defined as a “at least one of objects” concept (see Section 4.1). Finally, ODALISQUE has a standard navigation toolbar, with home, next and previous buttons. The logical constructors \top , \sqcap , \sqcup , \neg , \exists , and \geq are respectively displayed as Thing, and, or, not, some, and min.

5.2 Ontology

To illustrate the browsing of ontologies with a navigation scenario, we have created the ontology of the staff of our research laboratory, IRISA. This ontology includes the knowledge base Σ_{ex} of Section 2, and adds:

- 2 concepts *Man* and *Woman*, that are subsumed by *Person*;
- a concept *Theme*, that is the class of the research themes;
- a role *hastheme*, that represents relations from a team to its theme;
- the role *isthemeof*, that is the inverse of the *hastheme* role.

The A-Box now contains 182 persons, 29 research teams, 5 themes and 241 relations between these objects.

5.3 Navigation

The starting point of our scenario is characterized by the current query $q = \top$, the dynamic taxonomy showing only its root \top , and the extension of the current query containing

²<http://jena.sourceforge.net/>

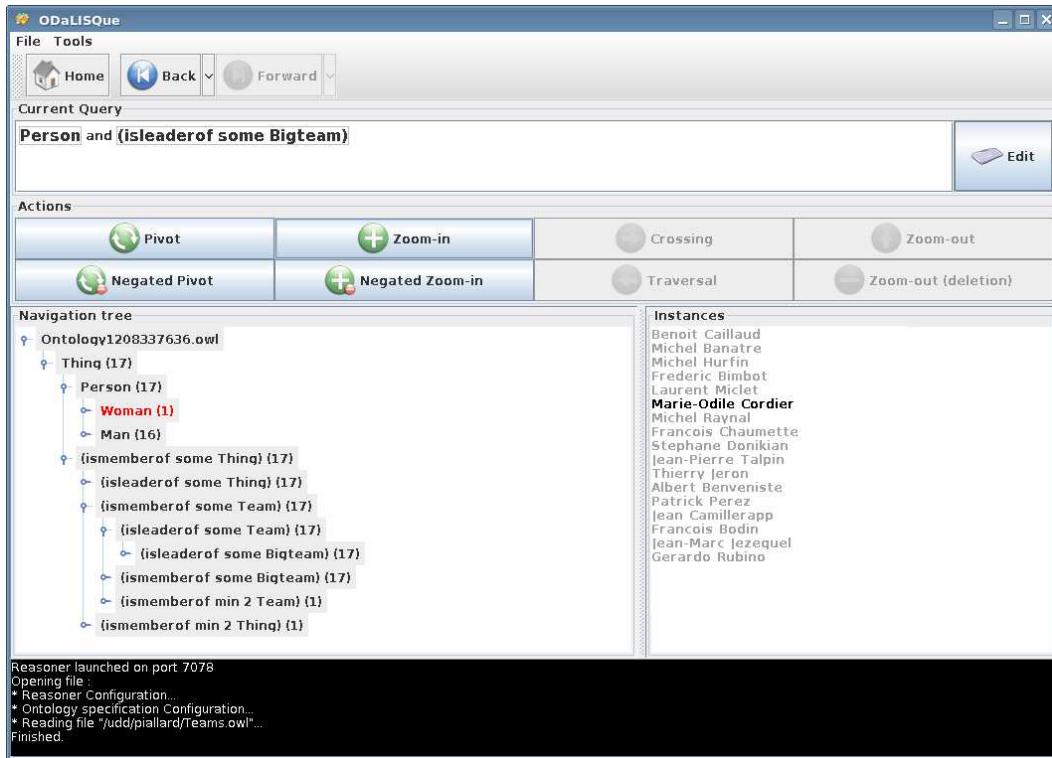


Figure 3. Screenshot of ODALISQUE

all the objects of the ontology. The following scenario illustrates the flexibility introduced by traversal and reversal links to build complex queries. It shows how a set of teams can be selected by switching between two points of view: the teams and their leaders.

5.3.1 Zooming in Teams

The first point of view we want to access is the teams with at least 5 members. The sub-increments of the increment \top are the concepts *Team*, *Person*, *Theme*, and the concepts $\exists r.\top$ for the roles *hasmember*, *hastheme* and their inverses. The sub-increments of the increment $\exists hasmember.\top$ are the concepts $\exists hasmember.Person$, $\exists hasleader.\top$, and $\geq 2 hasmember.\top$. By 4 successive increment openings, we can reach the increment $x = \geq 5 hasmember.Person$ from the increment $\geq 2 hasmember.\top$ (1 refining \top into *Person*, and the 3 others incrementing the cardinality from 2 to 5). The selection of this increment allows the following navigation links: zoom-in and pivot. Both navigation links replace the current query by $q = \{\geq 5 hasmember.Person\}$, and the extension is reduced from 216 objects to 13 teams. The dynamic taxonomy also changes: the sub-increments of \top are now only *Team*, $\exists hastheme.\top$, and $\exists hasmember.\top$,

i.e. relevant concepts for teams.

5.3.2 Traversing from Teams to Persons

The second point of view we want to access is the persons who lead the teams we have selected with the last query. With the selection of the increment $x = \exists hasleader.Person$, and the traversal link, the current query becomes $q = \{Person, \exists isleaderof.(\geq 5 hasmember.Person)\}$. The point of view has changed from teams to persons, from a set of 13 teams to the set of their 13 leaders. Hence, the dynamic taxonomy changes too: it contains only increments that are relevant to persons.

5.3.3 Zooming out Persons

From the definition of *Bigteam*, it can be demonstrated that every leader of team with at least 5 members is the leader of a big team, i.e. $\exists isleaderof.(\geq 5 hasmember.Person) \sqsubseteq \exists isleaderof.Bigteam$. Hence, the increment $\exists isleaderof.Bigteam$ can be used as a zoom-out link in order to expand the selected leaders. Then the current query becomes $q = \{Person, \exists isleaderof.Bigteam\}$, and the exten-

sion now contains 17 persons. Note here that we have expanded the set of leaders by expanding the set of teams through the role *isleaderof*. This shows that zoom-out works on complex concepts as well as on concept names.

5.3.4 Zooming in Persons

Next, the user may want to select only women among the leaders of big teams. The selection of the increment *Woman* (the screenshot of ODALISQUE at this time is given in Figure 1) and the navigation link of zoom-in gives a new query: $q = \{Woman, \exists isleaderof.Bigteam\}$. We notice that *Person* was replaced by *Woman*, because the statement $Woman \sqsubseteq Person$ holds. After this operation, the increment *Man* disappears.

5.3.5 Reversing back to Teams

Finally, the user wants to come back to the initial point of view, i.e. teams, but taking into account all previous navigation steps. So the last step to perform is following a reversal navigation link through the role *isleaderof*. This is done by selecting the concept $\exists isleaderof.Bigteam$, as an element of the current query q . This leads to the final query $q = \{Bigteam, \exists hasleader.Woman\}$, i.e. to the selection of big teams, whose leader is a woman.

From there, the user can access the themes or the members of this set of teams. He can then further refine this set of teams according to their themes and members by traversing the roles *hastheme* and *hasmember*.

6. Conclusion

In this paper, we have demonstrated how dynamic taxonomies can be adapted and applied to the browsing of web ontologies. This has been done for the OWL DL language, which is the most expressive among decidable ontology languages. A prototype, ODALISQUE, was developed. It is fully compliant with the OWL standard, and only assumes a non-empty A-Box. It relies on existing OWL reasoners, hence ensuring the correctness of its answers. It includes all features of dynamic taxonomies, and introduces new possibilities. First, concepts are not only names, but can be complex concepts made of various logical constructors. Complex concepts are not be only used in queries, but in dynamic taxonomy, which classical navigation modes can be applied (e.g., zoom-in, zoom-out, pivot). Second, relations can be defined between objects through role assertions, and this entails new navigation links for reversing a query (change of point of view), and traversing a role from a set of objects to a set of related objects. Third, because of the increased diversity of navigation links, it appears that

navigation links can be defined not only on increments, i.e. concepts in dynamic taxonomies, but also on objects in extensions, and on some query parts.

Our first experiments have revealed important improvements to be done. First, OWL reasoners are costly and strongly limit the size of ontologies we can browse. ODALISQUE could be made much more efficient either by implementing a dedicated reasoner or by pre-computing and caching many inference results. Second, ODALISQUE does not currently provide any means for editing an ontology. However, we think it can be done through the same interface (i.e., local views), as it has been done in Camelis for logical information systems. Third, many ontologies have no A-Box and are defined only at the terminological level. It would be interesting to provide similar browsing capabilities as in ODALISQUE for such ontologies. Finally, OWL is a logic that works under the open world assumption, which is often counter-intuitive when browsing real data. However, this can hardly be changed without breaking compliancy with OWL standards.

References

- [1] S. Ferré. CAMELIS: Organizing and browsing a personal photo collection with a logical information system. In J. Diatta, P. Eklund, and M. Liquière, editors, *Int. Conf. Concept Lattices and Their Applications*, volume 331 of *CEUR Workshop Proceedings ISSN 1613-0073*, pages 112–123, 2007.
- [2] S. Ferré and O. Ridoux. An introduction to logical information systems. *Information Processing & Management*, 40(3):383–419, 2004.
- [3] S. Ferré and O. Ridoux. Logical information systems: from taxonomies to logics. In *Int. Work. Dynamic Taxonomies and Faceted Search (FIND)*, pages 212–216. IEEE Computer Society, 2007.
- [4] R. Fikes, P. Hayes, and I. Horrocks. Owl-ql: A language for deductive query answering on the semantic web. Technical Report KSL 03-14, Stanford University, Stanford, CA, 2003.
- [5] I. Horrocks, P. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
- [6] I. Horrocks and U. Sattler. Ontology reasoning in the SHOQ(D) description logic. In B. Nebel, editor, *Int. Joint Conf. Artificial Intelligence*, pages 199–204. Morgan Kaufmann, 2001.
- [7] S. Jean, Y. A. Ameur, and G. Pierra. Querying ontology based database using ontoql (an ontology query language). In *OTM Conferences (1)*, pages 704–721, 2006.
- [8] G. Karvounarakis, A. Magkanaraki, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, and K. Tolle. Querying the semantic web with rql. *Computer Networks*, 42(5):617–640, 2003.
- [9] G. Sacco. Dynamic taxonomies: a model for large information bases. *Knowledge and Data Engineering, IEEE Transactions on*, 12(3):468–479, 2000.

- [10] G. M. Sacco. The intelligent e-sales clerk: the basic ideas. In M. R. et al, editor, *Int. Conf. Human-Computer Interaction (INTERACT)*, pages 876–879. IOS Press, 2003.