An Interactive Guidance Process Supporting Consistent Updates of RDFS Graphs

Alice Hermann, Sébastien Ferré, Mireille Ducassé

An Interactive Guidance Process Supporting Consistent Updates of RDFS Graphs

Alice Hermann[1], Sébastien Ferré[2], and Mireille Ducassé[1]

[1] IRISA, INSA Rennes, Campus de Beaulieu, 35042 Rennes cedex, France
[2] IRISA, Université de Rennes 1, Campus de Beaulieu, 35042 Rennes cedex, France

Abstract. With existing tools, when creating a new object in the Semantic Web, users benefit neither from existing objects and their properties, nor from the already known properties of the new object. We propose UTILIS, an interactive process to help users add new objects. While creating a new object, relaxation rules are applied to its current description to find similar objects, whose properties serve as suggestions to expand the description. A user study conducted on a group of master students shows that students, even the ones disconcerted by the unconventional interface, used UTILIS suggestions. In most cases, they could find the searched element in the first three sets of properties of similar objects. Moreover, with UTILIS users did not create any duplicate whereas with the other tool used in the study more than half of them did.

1 Introduction

Updating existing Semantic Web (SW) data is crucial to take into account information regularly discovered. This is, however, tedious and in practice data from the SW are rarely updated by users. In the Web 2.0, users, nevertheless, significantly contribute to the production of data, thus, motivation is not a problem. Models exist to bridge the gap between the SW and the Web 2.0, for example by linking tags created by users with SW vocabulary [13,11]. There are, however, still difficulties to integrate the Web 2.0 data in SW, for example to automatically align users tags and resources of the SW. Moreover, SW data are richer than user tags. SW indeed allows a more complex representation of data, as well as more elaborate queries. It is therefore important that users can directly create data in a SW format.

This paper presents UTILIS (Updating Through Interaction in Logical Information Systems), a method that uses existing objects and the current partial description of a new object, to help the user create that new object. UTILIS searches for objects similar to the object being created, namely objects having properties and values in common with it. These objects and their properties are used as suggestions to extend the description of the new object. In the following, examples and experiments are related to the extension of an annotation base of comics panels. An excerpt of the base is shown in Figure 1. Panel A is taken

```
Panel A
> collection : Ma vie à deux
> character : Missbean, Missbean's cat
> bubble : (a speech bubble,
said by Missbean, spoken to Missbean's cat)
```

```
Panel B
> collection : The Adventures of Tintin
> character : Tintin, Snowy
> bubble : (a speech bubble,
said by Tintin, spoken to Snowy)
```

```
Panel C
> collection : Peanuts
> character : Snoopy, Sally Brown
> bubble : (a speech bubble,
said by Sally Brown, spoken to Snoopy)
> bubble : (a thought bubble, said by Snoopy)
```

```
Panel D
> collection : Ma vie à deux
> character : MissBean, Babybean
> bubble : (a speech bubble,
said by MissBean, spoken to Babybean)
> bubble : (a speech bubble,
said by Babybean, spoken to Missbean)
```

Fig. 1. Excerpt of the comics annotation base

from collection *Ma vie à deux*, with *Missbean* and her cat as characters, it has a speech bubble said by Missbean to her cat.

The main contribution of this paper is an interaction process that helps users constitently create new objects by suggesting properties and values finely adapted to the very object being created. The already known properties of a new object are used to suggest others. Let us assume that a user annotates a new panel and specifies its collection, it may help to suggest characters, it is likely that this panel and those of the same collection have characters in common. The process uses a set of relaxation rules, inspired by the work of Hurtado et al. [9], and an efficient algorithm for computing suggestions. An advantage of our approach is that the definition of an ontology is not necessary to calculate the suggestions, although UTILIS may use an ontology to improve its suggestions when one is available.

A user study conducted with students shows that they have used the suggestions of UTILIS. They found them relevant. In most cases, they could find the searched item in the first three sets of suggestions. In addition, they have appreciated the suggestion mechanism, indeed 14 students out of the 18 wish to have it in a SW data editor. Even if some users were disconcerted, the base resulting from the use of UTILIS was more consistent and contained less errors than when using Protégé.

Section 2 gives definitions related to the SW languages. It introduces logical information systems which are used to interact with users. Section 3 specifies our approach, UTILIS. Section 4 presents the creation of a panel description . Section 5 presents the user study. Section 6 compares our approach to related work.

2   Preliminaries

RDF, RDFS and SPARQL. RDF and RDFS are Semantics Web languages that enable tools to be interoperable. The basic elements of these languages are resources and triples. A resource can be either a URI (absolute name of a resource),

English	Which are the panels with at least one bubble said by Missbean?
SPARQL query	SELECT ?x WHERE {?x a :panel . ?x :bubble ?y . ?y :saidBy \<MissBean\>)}

Fig. 2. A question and its translation in SPARQL

a literal or an anonymous resource. A triple consists of 3 resources. A triple *(s, p, o)* can be read as a sentence where *s* is the subject, *p* is the verb, called predicate, and *o* is the object.

RDF allows data to be represented. For example, in the annotation base, triple (*<PanelK>, :character, <Missbean>*) can be read as "Panel K has character Missbean". RDF has a predefined vocabulary to represent membership in a resource class (*rdf:type*), the hierarchy between classes (*rdfs:subClassOf*) and between the properties (*rdfs:subPropertyOf*). For example, triple (*:Bubble1, rdf:type, :SpeechBubble)* tells that "Bubble1 has type SpeechBubble" or simply " Bubble1 is a speech bubble". Resource :*SpeechBubble* is a class. Triple (:*SpeechBubble, rdfs:subClassOf, :Bubble)* tells that "the SpeechBubble class is a subclass of Bubble", or "Each speech bubble is a bubble." In addition, RDFS is a language of knowledge representation with inference power [8]. By inference, the latter two triples can be used to deduce the previous triple (*<Bubble1>, rdf:type, :Bubble)*. In the following, for readability reasons, descriptions of the objects are written in Turtle notation [1]. For example, the following triples describing a new panel ((*<PanelK>, rdf:type, Panel) (<PanelK>, :character, <Missbean>)*) are written in Turtle as (*<PanelK> a panel; :character <Missbean>*).

SPARQL is a query language for RDF based on *graph pattern matching* [14]. A question and its translation into SPARQL are shown in Figure 2.

Logical Information Systems (LIS). LIS [6] are a paradigm of information retrieval and exploration, combining querying and navigation. They are close to the paradigm of faceted search [15]. Their query language has an expressiveness similar to that of SPARQL, and a syntax similar to Turtle [5]. A prototype, Sewelis[1], has been implemented. The user navigates from query to query. Navigation links are automatically computed from the dataset, and suggested to users, in a way that ensures that guided navigation is safe (no dead-end), and complete (every query that is not a dead-end can be reached). UTILIS is implemented in Sewelis.

3 UTILIS : An Interactive Guidance

This section describes UTILIS, our interactive guidance method to help users create objects in an RDFS graph. UTILIS searches for objects similar to the description of a new object, i.e., objects having common properties and values. Figure 3 illustrates the interactive process to refine the description of a new
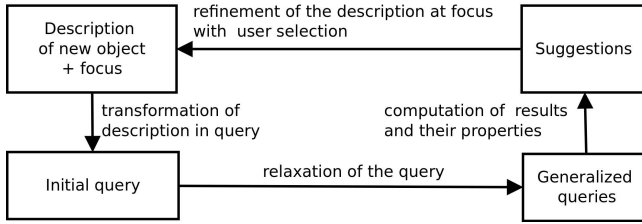
---

[1] http://www.irisa.fr/LIS/softwares/sewelis

Fig. 3. Interactive refinement of the description of a new object

Description	<PanelK> a :panel; :collection <Ma vie à deux>; :character [ ]
Initial query	SELECT ?z WHERE {?x a :panel . ?x :collection ?y . ?x :character ?z . FILTER (?y = <Ma vie à deux>)}

Fig. 4. A description and its transformation into the initial query

object. The initial query is obtained from the current description of a new object and from a focus which is an element of the description that the user wants to complete. This description is transformed into a query (Section 3.1). That query is used as a starting point to obtain generalized queries using relaxation rules (Section 3.2). The initial and generalized queries allow similar objects to be retrieved. Those objects and their properties are used as suggestions to complete the new description (section 3.3). After refinement, the new description will be used to define a new query and so on. An efficient algorithm to compute the suggestions from the RDFS graph and the current description of the new object has been implemented with dynamic programming techniques (section 3.4).

3.1 Transformation of the Description into the Initial Query

The description consists of the set of elements that have been entered by the user on the new object. Figure 4 shows a description and the initial query obtained after transformation. The description means that panelK is a panel from the *Ma vie à deux* Collection, it has characters which have not yet been specified. The focus is represented by the underlined part. The description is transformed into a query in three steps. Firstly, the identity of the new object is replaced by a variable. For example, <PanelK> is replaced by $?x$. Secondly, the description is transformed into a SPARQL *graph pattern* in order to have only one modifiable component per triple. Each triple is composed of a single individual. The other elements of the triple are variables or predefined property *rdf:type*. For example, (:collection <Ma vie à deux>) is transformed into (*?x :collection ?y FILTER (?y = <Ma vie à deux>)*). Finally, the description is transformed into a query by setting the variable at the focus as the variable to look for, and the body of the query matches the graph pattern of step 2. This initial query provides all the individuals that have the already known properties and values of the new object.

Rule	Initial triple or constraint	Relaxed triple	Condition
SuperProperty	$?x$ p_1 $?y$	$?x$ p_2 $?y$	p_1 `subp` p_2
SuperClass	$?x$ `a` c_1	$?x$ `a` c_2	c_1 `subc` c_2
Resource	$?x = r_2$	`nil`	
Property	$?x$ p_1 $?y$	`nil`	$\nexists p \neq p_1.(p_1$ `subp` $p)$
Class	$?x$ `type` c_1	`nil`	$\nexists c \neq c_1.(c_1$ `subc` $c)$

Fig. 5. Relaxation rules. Variables have a leading question mark, r_i are resources, p_j are properties, and c_k are classes; a relaxed triple set to `nil` means that the initial triple is actually removed. `type`, `subc`, `subp` are abbreviations for `rdf:type`, `rdfs:subClassOf` and `rdfs:subPropertyOf`

3.2 Query Relaxation

After adding some property-value pairs, the description of a new object, however, becomes unique in the database, because the database objects do not all have the same property-value pairs. The initial query, obtained from the description of the new object, then leads to no results. In order to continue to offer suggestions to the user, UTILIS seeks objects similar to the new object by generalizing the query. To generalize the initial query, we have defined relaxation rules, inspired by the query approximation rules of Hurtado et al. [9]. Figure 5 shows the set of relaxation rules, that apply to triples. The first column shows the rule name, the second column shows the triple before relaxation, the third column shows the relaxed triple and the fourth column shows potential conditions for the application of the rule. Except the rules *SuperProperty* and *SuperClass*, the rules do not depend on an ontology. Rule *SuperProperty* applies to a triple with a variable as subject, another one as object and a resource p_1 as predicate, under the condition that p_1 is a subproperty of another property p_2. After applying that rule, the subject and object remain the same but property p_1 is replaced by property p_2. For example, (`?x, :spokenTo, ?y`) can be relaxed in (`?x, :inConversationWith, ?y`) by the application of that rule if there exists triple (`:spokenTo, rdfs:subPropertyOf, :inConversationWith`). Relaxed triple `nil` corresponds to the suppression of the initial triple. The distance between the original query and a generalized query is the number of rules applied to switch from one to the other.

At distance 0, are the results of the initial query, namely the ones obtained directly from the description without generalization. For example, let us suppose that the initial query is (`SELECT ?x WHERE {?x a :panel . ?x :collection ?y . ?x :character ?z. ?x :character ?a . FILTER (?y = <Ma vie à deux> && ?z = <MissBean> && ?a = <Fatbean>)}`) [2]. A generalized query can be generated without the constraint (`?z = <Missbean>`) by applying rule *Resource* on this constraint. That generalized query is at distance one. At the same distance, rule *Resource* can also be used on another constraint,

---

[2] What are the panels of *Ma vie à deux* with characters Missbean and Fatbean ?

for example on constraint ($?a$ = <Fatbean>). The union of all the results of generalized queries produced by a single relaxation step are the proposed results at distance one. Rules can be combined. The order of rule applications has no impact on the generalized queries. This confluence property allows an efficient algorithm using dynamic programming to be implemented (Section 3.4).

3.3   Refinement of the New Object Description

Similar objects are the results of queries, generalized or not. Those objects and their properties are used as suggestions to complete the new description. The suggestions are resources, classes or properties. If the focus is on a resource, the suggestions are classes and properties. A powerful feature of UTILIS is that users can add features to a description at any time and also at any "point" of the description. For example, a user can decide to add an information to the description of any panel already annotated, attaching that information to any existing individual. The place to be extended has to be identified. This is the aim of the focus. In order to avoid that users have to explicitly specify the focus at every step, the default strategy is as follows. When adding a property to the description of the object, the focus is moved to the variable representing the value of the property because it is assumed that the user will most probably want to select or enter a value there after. When the value of the property has been given, the focus returns to the description of the object. When adding a class to the description of the object, the focus remains on the description of the object.

To present suggestions to the user and allow him to complete the description of the new object, the interaction mechanisms of Sewelis, initially dedicated to navigation, have been reused to support creation. By default, only the suggestions at the smallest distance are proposed, the list can be enlarged to bigger distances upon user's request, until all objects compatible with the searched element are proposed. At any time, users can manually enter what they want to add. That is necessary for the new values. Auto-completion mechanisms help find existing values. Once a suggestion is selected by the user, the new description is used to define the initial query that will be generalized to make new suggestions to the user.

3.4   Relaxation Algorithm

A naive algorithm to compute the results at distance d of query q could consist in generating all generalized queries at distance d, and computing the union of their results. The generalized queries at distance d would be obtained by applying d relaxations on the n triples of the query, which amount to $\binom{n}{d}$ generalized queries. Due to the nature of queries whose graph patterns are trees because they are derived from Turtle expressions, the results of each generalized query can be computed in $O(n)$ set operations (intersections and relation crossings). The cost of the naive algorithm is too expensive, especially as the generalized queries are only intermediate steps in our approach.

In UTILIS, we use a more efficient algorithm implemented in dynamic programming. Tabled function $E(d, D)$ directly computes results at distance d of the generalizations of query *SELECT ?x WHERE { ?x D}*, where D is a Turtle description without subject (list of predicate-object pairs). To make this query correspond to the initial query, it is sufficient to define D as the rewriting of the new object description from the focus. For example, for the initial description *<PanelK> a :panel; :collection [[]*, D is equal to *is :collection of [a :panel]*. Function E is defined by recursively calling itself with smaller distances and/or sub-descriptions. The base cases are when $d = 0$ and when D is an atomic description (i.e, of the form *a c*, *p r* or *p []*). The algorithm is defined by a set of equations, covering all combinations of a distance and of a description. For example, the equation that defines E for the conjunctions of descriptions is

$$E(d, D_1; D_2) = \bigcup_{i=0}^{d}(E(i, D_1) \cap E(d - i, D_2)).$$

This equation says that a result for a description of the form $D_1; D_2$ is both a result at distance d_1 from D_1 and a result at distance d_2 from D_2, such that $d_1 + d_2 = d$. Distance d is distributed between the two sub-descriptions ($d_1 = i$ and $d_2 = d - i$) in every possible way for $i = 0..d$. The equations for the description of the form $p [D_1]$ and *is p of* $[D_1]$ are similar, using the traversal of a relationship instead of an intersection. The results at distance d for classes (resp. properties) are based on the superclasses (resp. superproperties) at distance d in the hierarchy of classes (resp. properties).

The intermediate results of $E(d, D)$ are stored in a table with a line for each distance from 0 to d ($O(d)$ lines), and a column for D and each sub-description of D ($O(n)$ columns). The complexity for computing a cell of this table is $O(d)$ set operations. Therefore, the complexity of our algorithm is $O(nd^2)$ set operations, namely, polynomial instead of combinatorial for the naive algorithm.

4   Example

To illustrate UTILIS, this section describes some details of the creation steps of the description of a new panel in the annotation database. The database contains the panels shown in Figure 1 and six other panels. Let us assume that a user wants to add the panel of Figure 6, named PanelK. It is part of the *Ma vie à deux* collection. It has two characters, Missbean and Fatbean, and a speech bubble said by Missbean to Fatbean.

Figure 7 shows steps 2-7 of the creation. At each step, the top box contains the current description of the object, the focus is underlined, the bottom box contains suggestions with, in front of each of them, the minimum distance between the initial query and the generalized query which led to this suggestion. For space reasons, at each step we show only a limited number of suggestions. The element in boldface corresponds to the choice of the user.

At step 2, the current description is *<PanelK> a :panel*. Suggestions adapt to that description: they are the properties of at least one panel. The user chooses

Fig. 6. Panel of *Ma vie à deux : Pour le meilleur et pour le pire !*, by *Missbean, City Editions*

:collection []. The upper part of Figure 8 shows the user interface for step 3, annotations in black have been made by hand, the current description is on the left side. On the right side, suggested resources for the focus are listed, here the collections. Above that area, the number of suggestions is indicated and a *More* button allows them to be expanded to larger distances containing results. At step 3, the user chooses *<Ma vie à deux>* among all the collections. Let us go back to Figure 7, at step 4 the description is *<PanelK> a :panel; :collection <Ma vie à deux>*. The suggestions are the properties of at least one panel of *Ma vie à deux*. The property selected by the user is *:character []* to specify characters of the new panel. At step 5, the suggestions are all the characters of the *Ma vie à deux* panels already annotated. The user selects *<Missbean>* and *<Fatbean>*. At step 6, the initial query, corresponding to the description, has no results. Indeed, those two characters do not appear together in any panel of the base yet. By relaxation, UTILIS makes suggestions to the user at distances 1 and 7. The lower part of Figure 8 shows the user interface of step 7, suggested classes and properties are in the middle part. From there on, the user can continue the description of the panel and when he decides that it is complete, he adds it to the base with the *Assert* button.

5   Usability Study

In order to assess the suggestions made by UTILIS during the creation of new objects, an experiment was carried out. Users have tested and evaluated the usability of UTILIS and Protégé [12][3].

The interface and guidance of Protégé are representative of the current editors. Indeed, two thirds of the Semantic Web users use it as editor [3]. Protégé requires the domain and range of properties to be defined in order to provide any suggestions. When a user chooses the class of an individual, a form with the properties having this class as domain is created. For the values, the suggestions are individuals of the range of each property.

---

[3] `http://protege.stanford.edu`

Step 2

```
<PanelK> a :panel

0 − :bubble []
0 − :character []
0 − :collection []
1 − :said by []
1 − :spoken to []
```

Step 3

```
<PanelK> a :panel;
:collection []

0 <Ma vie à deux>
0 <Peanuts>
0    <The adventures of
Tintin>
0 <Uncle Scrooge>
```

Step 4

```
<PanelK> a :panel;
:collection <M>

0 :bubble []
0 :character []
3 :said by []
3 :spoken to []
```

Step 5

```
<PanelK> a :panel;
:collection <M>;
:character []

0 <Missbean>
0 <Babybean>
0 <Fatbean>
0 <Missbean's cat>
1 <Donald Duck>
```

Step 6

```
<PanelK> a :panel;
:collection <M>;
:character <MB>, <FB>

1 − :bubble []
7 − :said by []
```

Step 7

```
<PanelK> a :panel;
:collection <M>;
:character <MB>, <FB>;
:bubble []

1 a :speechBubble
1 :said by []
1 :spoken to []
3 a :thoughtBubble []
```

Fig. 7. Creation steps for new panel PanelK : It is related to the *Ma vie à deux* collection. It has characters Missbean and Fatbean. It has a speech bubble said by Missbean spoken to her husband : Fatbean.

5.1   Methodology

The subjects consisted of 18 master students in computer science. They had prior knowledge of relational databases, but knew neither Sewelis, nor Protégé, nor the Semantic Web. For each editor, they had to perform the same tasks. The experiment procedure for each subject was the following: 1) read an introductory note on the overall experiment ; 2) learn how to use an editor with a tutorial of 30 minutes with an example of annotation creation ; 3) develop and create annotation with the editor for 30 minutes ; 4) complete a questionnaire about the editor ; 5) proceed by repeating steps (2) to (4) with the other editor, and 6) complete a comparison questionnaire.

The subjects were asked to update an existing database, describing comics panels. The base was identical at each start of a test session. It consisted of 362 individuals, divided into 16 classes and connected by 20 properties, including 89 panels. Under Sewelis and Protégé, the class and property hierarchies were the same. The subjects were to update the database with two sets of panels. They were divided into four groups, each group conducted two sessions of 1:30. Each group tested one of the two editors on one of the two panel sets (set1 or set2) then it tested the other editor with the other panel set. Each set consisted of 11 panels including the panel used for the tutorial and 10 panels to be annotated, with at least 2 panels from the same collection. The panels were presented on paper in the same order for all subjects. They already existed in the database, but had no description. Subjects were instructed to enter a maximum of information on each panel, taking the description of the existing panels as model. They were

Fig. 8. UTILIS screen shots of steps 3 and 7 of PanelK creation

Table 1. Average number of annotated panels and pourcentage of average number of errors, under Protégé and UTILIS, according to the tools and set orders

	Panels under Protégé	Panels under UTILIS	Errors under Protégé	Errors under UTILIS
groups UTILIS-first	9.6	9	3.5%	4.9%
groups Protégé-first	8.8	8.6	5.4%	6.7%
groups set1-first	8.7	8.6	4.1%	5.2%
groups set2-first	9	8.5	4.9%	6.6%

instructed to reuse existing information as much as possible. Nevertheless, they could create new individuals when needed.

During the experiments, a log file of user actions was created, in particular it recorded the number of descriptions created in Sewelis and Protégé, and for Sewelis, the mode of selection of description items and the number of enlargements of suggestions.

5.2   Results

This section presents the results of the experiment. In particular, the nature of the errors made by users respectively under UTILIS and Protégé is commented. The objective and subjective reactions of users with respect to the suggestions are discussed.

No difference between the groups. As illustrated in Table 1, there are no significant differences in the number of created descriptions (9 in average) between users of the four groups. There are also no significant differences in the average number of errors related to the total number of added triples. The order in which tools were used and the order of the sets had, thus, no influence. As a consequence, in the following the results are globally reported.

Table 2. Percentages of elements of description selected in suggestions, found by completion and created, together with the range of suggestion enlargements and the percentage of elements selected in the first three sets of suggestions

	Properties and classes	"Collections"	"Characters"	"Speakers"	"Recipients"	"Places"	"Other elements"
Selection in suggestions (%)	91	91	90	89	82	58	21
Auto-completion (%)	9	9	8	11	15	25	21
Creation (%)	0	0	2	0	3	17	58
Selection in first 3 sets (%)	91	91	92	89	84	68	44
Range of enlargements	0-4	0-1	0-2	0-1	0-3	0-4	0-6

Suggestions were relevant. UTILIS suggestions have been used by users during the creation of new panels. Table 2 shows the proportion of description elements chosen by selecting a suggestion, by auto-completion and by creation. It also shows the range of suggestion enlargements and the percentage of suggestions selected in the first three sets of suggestions. The three modes of selection have been used. One can see, for example, that adding characters was made in 90% of the cases by selecting a suggestion, in 8% by auto-completion and in 2% by creation. The elements related to the collection, to the characters and to bubble interlocutors were, if they existed, selected more than 80% of the cases in the first 3 sets of suggestions. For locations and other elements, the suggestions were less useful. These descriptive elements are more subjective than the above mentioned ones and they often have been created by users. The questionnaires completed after the experiments give a consistent view. Indeed, 16 subjects found suggestions relevant for characters, 15 for the collection, 12 for interlocutors, but only 6 for the locations and 3 for other elements.

Suggestions were appreciated. Subjects were asked which of the elements of each editor they would like to see in an ideal editor. Ten subjects wished to have the properties in a form as in Protégé. From UTILIS, 14 subjects wished to retain the suggestions adapted to the object being created, and 11 subjects wanted the auto-completion search mode. Nine (resp. 7) subjects wished to have UTILIS suggestions for some of (resp. all) properties.

Consistency was better ensured under UTILIS. Table 3 shows the number of errors per error type introduced in the base by the users, respectively under Protégé and UTILIS. The number of users who made the errors is given between parentheses. The most important result is that 27 duplicates were introduced in the base under Protégé, while none was created under UTILIS. A duplicate is a new individual, created whereas an existing one would have been relevant. Furthermore, under Protégé 50% more wrong values were introduced than under

Table 3. Number of errors (number of users who made them) per error type in the base, under Protégé and UTILIS

	Duplicates	Wrong values	Forgotten elements	Cancelled insertions	Wrong format	Wrong properties	Misplaced focus	Total
Protégé	**27 (11)**	**46 (10)**	33 (11)	39 (10)	**0 (0)**	**0 (0)**	**0 (0)**	145 (18)
UTILIS	**0 (0)**	**30 (9)**	33 (3)	30 (9)	**15 (1)**	**48 (3)**	**22 (9)**	178 (16)

UTILIS. It should also be noted that the duplicates have been created by more than half of the users (11 subjects) under Protégé. Moreover, 9 of the 18 users did not introduce any duplicate or value errors at all in the base under UTILIS against 2 only under Protégé. Consistency was thus better ensured with UTILIS. Both the suggestion and auto-completion mechanisms of UTILIS help users to find existing individuals without having to browse through entire lists as in Protégé. As a matter of fact, 13 subjects reported being bothered to have to traverse the entire lists of individuals in Protégé. Furthermore, suggestions and auto-completion provide examples of formats and types of similar cases. It helps users create new individuals that are consistent with existing individuals. Note that the identification and reuse of resources is what distinguishes five stars linked open data (LOD)[4] from four stars LOD, according to the star scheme of LOD.

More handling errors under UTILIS but made by few users. Table 3 shows other types of errors. Cancelled elements are elements which were not committed, or were deleted and not added afterwards. Both the number of cancelled and forgotten elements are similar under both tools. However, under UTILIS the number of users who forgot elements is small (3 users) and significantly lower than under Protégé. Forgotten elements under Protégé are individuals for which several occurrences of the same property were needed, for example several characters in a given panel. There are three types of error made only under UTILIS. Wrong types is a format error, for example the creation of a literal instead of an individual. Examples of wrong properties are using a superproperty or using a property in the wrong direction. A misplaced focus results in an extension attached to a wrong place. These errors cannot occur under Protégé, the forms to be filled are static. It should be noted that all the wrong types have been introduced by a single user and the wrong properties by only 3 users. Regarding the misplaced focus, Section 3.3 presented the default strategy for focus positioning. Users may want to fill in the descriptions in a different order than the default strategy. One objective of this study was to observe how users would cope with focus positioning. Half (9) of the users made focus errors but most (7) of them made that mistake only once. The error occurred at any time, not necessarily

---

[4] `http://www.w3.org/DesignIssues/LinkedData.html`

while describing the first panels. Once the error made, seven users thus did not repeat it. In the end, only two users were still abashed by the focus positioning. Altogether, two users managed to make no error at all under UTILIS whereas none did under Protégé. In conclusion, few users were still disconcerted by the unconventional user interaction model of UTILIS at the end of the session. We conjecture that those difficulties came from the extra flexibility of UTILIS. In further work we will study how to help those users manage with the flexibility of UTILIS.

6   Related Work

Editors for Semantic Web data can be dedicated tools (e.g., Protégé [12], OKM (OKM Ontology Management) [4], Gino [2], QuiKey [7]) or integrated into Wiki environments (e.g., SMW (Semantic Media Wiki) [17], KiWI [16], ACEWiki [10]). Their interface can be based on forms (e.g., Protégé, OKM, SMW), on natural language (e.g., Gino, ACEWiki), or only allow for the creation of one triple (or semantic link) at a time (e.g., KiWI, Quikey). We discuss in the following the key differences with UTILIS.

UTILIS does not need any preliminary preparation, nor any schema. Protégé requires the definition of domains and ranges to make any suggestion, because forms are derived from that information. The Semantic Forms[5] of SMW have to be defined manually. Editors based on natural language require the definition of a lexicon, and its relation to an ontology. Like OKM and Quikey, UTILIS can be applied from scratch on any RDFS graph.

UTILIS suggestions are based on individuals rather than on ontology axioms. All above editors, except OKM and Quikey, only look at ontology axioms, i.e. the schema, to compute suggestions, generally starting from the class of the new object. This implies that, at step 5 of our scenario, Protégé lists all characters in alphabetical order, whatever the existing objects in the base. With UTILIS, characters that are already linked to a panel in the base would be suggested first; and if the collection of the panel has already been specified, then characters of that collection are suggested before other characters.

UTILIS suggestions depend on the full description of the new object. The flexible forms of OKM only depend on, and require, the class of the new object, and Quikey simply suggests (through auto-completion) all existing properties and individuals. UTILIS can provide suggestions from the start, when nothing is known about the new object, and refine them whenever an information is added to its description. At step 2 of our scenario, knowing the class improves the suggestion of properties. At step 5, knowing the collection of the panel improves the suggestion of characters. This works for description elements at arbitrary depth, e.g., for suggesting the locutor of a bubble of the panel.

---

[5] `http://www.mediawiki.org/wiki/Extension:Semantic_Forms`

UTILIS suggestions are available before any user input. All editors, except Protégé and ACEWiki, require the user to enter a few letters in order to get suggestions of individuals (e.g., characters). This can be explained by the fact that suggestions are not fine-tuned to the new object, and the alternative is then to list all instances of some class, like in Protégé. UTILIS can suggest short lists of individuals as soon as the description becomes specific. For long lists, UTILIS also provides auto-completion, similarly to Quikey.

7   Conclusion

We propose a method, UTILIS, which guides users through the creation of objects in an RDFS graph. The guidance takes benefit of existing objects and of the current description of a new object. At each step, the current description is used to find similar objects, whose properties are then used as suggestions to complete the description. Similar objects are the results of queries that are generalized from the current description. These queries are obtained by relaxation rules. An efficient algorithm to compute suggestions has been designed and implemented in Sewelis.

Compared to other RDFS editors, UTILIS suggestions are based on existing objects, rather than only on the RDF schema. They are, therefore, well adapted to each object being created. One advantage is that UTILIS does not need an ontology to be defined, although it can use one to improve its suggestions when one is available. An experiment has shown that subjects found the suggestions useful and they actually used them. In most cases, they could find the desired item in the first three sets of suggestions. Even if some users were disconcerted, the base resulting from the use of UTILIS was more consistent and contained less errors than when using Protégé. In addition 14 of the 18 subjects wanted to keep UTILIS suggestion mechanism in an editor.

References

1. Beckett, D., Berners-Lee, T., Prud'hommeaux, E.: Turtle - Terse RDF Triple Language. W3C Recommendation (January 2010)
2. Bernstein, A., Kaufmann, E.: GINO - A Guided Input Natural Language Ontology Editor. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 144–157. Springer, Heidelberg (2006)
3. Cardoso, J.: The semantic web vision: Where are we? IEEE Intelligent Systems, 84–88 (2007)

4. Davies, S., Donaher, C., Hatfield, J.: Making the Semantic Web usable: interface principles to empower the layperson. Journal of Digital Information 12(1) (2010)
5. Ferré, S., Hermann, A.: Semantic Search: Reconciling Expressive Querying and Exploratory Search. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 177–192. Springer, Heidelberg (2011)
6. Ferré, S., Ridoux, O.: An introduction to logical information systems. Information Processing & Management 40(3), 383–419 (2004)
7. Haller, H.: QuiKey – An Efficient Semantic Command Line. In: Cimiano, P., Pinto, H.S. (eds.) EKAW 2010. LNCS, vol. 6317, pp. 473–482. Springer, Heidelberg (2010)
8. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. CRC Press (2009)
9. Hurtado, C.A., Poulovassilis, A., Wood, P.T.: Query relaxation in RDF. J. Data Semantics 10, 31–61 (2008)
10. Kuhn, T.: How controlled english can improve semantic wikis. In: Semantic Wiki Workshop (SemWiki), vol. 464. CEUR-WS.org (2009)
11. Limpens, F., Gandon, F., Buffa, M.: Sémantique des folksonomies: structuration collaborative et assistée. In: Ingénierie des Connaissances (IC), pp. 37–48. Presses Universitaires de Grenoble (2009)
12. Noy, N.F., Sintek, M., Decker, S., Crubezy, M., Fergerson, R.W., Musen, M.A.: Creating semantic web contents with protege-2000. IEEE Intelligent Systems 16(2), 60–71 (2001)
13. Passant, A., Laublet, P.: Meaning of a tag: A collaborative approach to bridge the gap between tagging and linked data. In: Workshop Linked Data on the Web (LDOW). CEUR-WS (2008)
14. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C Recommendation (2008)
15. Sacco, G., Tzitzikas, Y. (eds.): Dynamic Taxonomies and Faceted Search: Theory, Practice, and Experience. The Information Retrieval Series, vol. 25. Springer, Berlin (2009)
16. Schaffert, S., Eder, J., Grünwald, S., Kurz, T., Radulescu, M., Sint, R., Stroka, S.: Kiwi - a platform for semantic social software. In: Semantic Wiki Workshop (SemWiki). CEUR-WS.org (2009)
17. Völkel, M., Krötzsch, M., Vrandecic, D., Haller, H., Studer, R.: Semantic Wikipedia. In: International conference on World Wide Web (WWW), pp. 585–594. ACM Press (2006)