Assisting the RDF Annotation of a Digital Humanities Corpus Using Case-Based Reasoning

Nicolas Lasolle[1,2(✉)], Olivier Bruneau[1], Jean Lieber[2], Emmanuel Nauer[2], and Siyana Pavlova[2]

[1] Université de Lorraine, CNRS, Université de Strasbourg, AHP-PReST, 54000 Nancy, France
{nicolas.lasolle,olivier.bruneau}@univ-lorraine.fr
[2] Université de Lorraine, CNRS, Inria, LORIA, 54000 Nancy, France
{nicolas.lasolle,jean.lieber,emmanuel.nauer,siyana.pavlova}@loria.fr

Abstract. The Henri Poincaré correspondence is a corpus composed of around 2100 letters which is a rich source of information for historians of science. Semantic Web technologies provide a way to structure and publish data related to this kind of corpus. However, Semantic Web data editing is a process which often requires human intervention and may seem tedious for the user. This article introduces RDFWebEditor4Humanities, an editor which aims at facilitating annotation of documents. This tool uses case-based reasoning (CBR) to provide suggestions for the user which are related to the current document annotation process. These suggestions are found and ranked by considering the annotation context related to the resource currently being edited and by looking for similar resources already annotated in the database. Several methods and combinations of methods are presented here, as well as the evaluation associated with each of them.

Keywords: Semantic Web · Content annotation · Case-based reasoning · RDF(S) · SPARQL query transformation · Digital humanities · History of science · Scientific correspondence

1 Introduction

Born in Nancy, France, in 1854, Jules Henri Poincaré is considered one of the major scientists of his time. Until his death in 1912, he relentlessly contributed to the scientific and social progress. Most known for his contribution in mathematics (automorphic forms, topology) and physics (3-Body problem resolution), he also played a significant role in the development of philosophy. His book *La Science et l'Hypothèse* [15] had a major international impact for the philosophy of science.

His correspondence is a corpus composed of around 2100 letters which includes sent and received letters. It gathers scientific, administrative and private correspondence which are of interest for historians of science. The *Archives*

J. Z. Pan et al. (Eds.): ISWC 2020, LNCS 12507, pp. 617–633, 2020.
https://doi.org/10.1007/978-3-030-62466-8_38

Henri-Poincaré is a research laboratory located in Nancy, in which one of the important works is the edition of this correspondence. In addition to the physical publication, a keen interest is devoted to the online publishing of this correspondence. On the Henri Poincaré website[1] are available the letters associated with a set of metadata. Different search engines may be used by historians or, more globally, by those who show an interest in the history of science. This has been achieved by the use of Semantic Web technologies: RDF model to represent data, RDFS language to represent ontology knowledge and SPARQL language to query data. During the human annotation process, several difficulties have emerged. Indeed, the annotation is a tedious process which requires the constant attention of the user to avoid different kinds of mistakes. The *duplication mistake* is encountered when the user inserts data that is already in the database. The *ambiguity mistake* happens when the user does not have enough information to distinguish items. It occurs when the same description or label is used for different items or when an item identifier does not give explicit information about its type and content. For instance, if a search is made based on the string `"Henri Poincaré"`, different types of resources may be returned. Indeed, the most plausible expected answer should refer to the famous scientist, but this term also refers to different institutes, schools and, since 1997, a mathematical physics prize exists named in his memory. The *typing mistake* is encountered when the user wants to write an existing word to refer to a specific resource but inadvertently mistypes it. If not noticed, this error can lead to the creation of a new resource in the database instead of referring to an existing resource. In addition to these possible mistakes, the cognitive load effect associated with the use of an annotation system should not be neglected. Depending on the volume of the corpus to annotate, this process could be a long-term project. Keeping the users motivated when performing the associated tasks is a real issue.

This article intends to present an efficient tool currently in use for content annotation. A suggestion system is proposed to the user to assist her/him during the annotation process. Four versions of the system have been designed: the *basic editor*, the *deductive editor*, the *case-based editor* and the last version, the *combination editor* which is a combination of the methods used in the two previous versions of the system. The last two versions use case-based reasoning (CBR) to find resources presenting similarities with the one currently being edited and thus take advantage of the already indexed content. The following hypotheses are made and are evaluated in the evaluation section:

Hypothesis 1 the use of CBR improves the suggestion list provided to the user.
Hypothesis 2 the combination of the use of CBR and RDFS entailment improves the suggestion list with respect to the use of CBR alone and of RDFS entailment alone.

Section 2 shortly introduces Semantic Web notions that are considered in this article, and presents a brief reminder of CBR. Section 3 explains the current

[1] http://henripoincare.fr.

infrastructure related to the Henri Poincaré correspondence edition and summarizes the previous work about the annotation tool. Section 4 focuses on how the use of CBR improves the annotation process and addresses the issues mentioned above. Section 5 describes the evaluation. Section 6 details some of the choices that have been made through the development of this editor and situates this work by comparing it with related works. Section 7 concludes and points out future works.

2 Preliminaries on Semantic Web Technologies and CBR

This section introduces the CBR methodology and terminology. A brief reminder of the Semantic Web notions and technologies that are RDF, RDFS and SPARQL is provided afterwards.

2.1 CBR: Terminology and Notation

Case-based reasoning (CBR [17]) aims at solving problems with the help of a *case base* $\mathtt{CB}$, i.e., a finite set of cases, where a case represents a problem-solving episode. A case is often an ordered pair $(\mathtt{x}, \mathtt{y})$ where $\mathtt{x}$ is a problem and $\mathtt{y}$ is a solution of $\mathtt{x}$. A case $(\mathtt{x}^s, \mathtt{y}^s)$ from the case base is called a *source case* and $\mathtt{x}^s$ is a *source problem*. The input of a CBR system is a problem called the *target problem* and is denoted by $\mathtt{x}^{\mathtt{tgt}}$.

The 4 Rs model decomposes the CBR process in four steps [1]. (1) A $(\mathtt{x}^s, \mathtt{y}^s) \in \mathtt{CB}$ judged similar to $\mathtt{x}^{\mathtt{tgt}}$ is selected (*retrieve step*). (2) This retrieved case $(\mathtt{x}^s, \mathtt{y}^s)$ is used for the purpose of solving $\mathtt{x}^{\mathtt{tgt}}$ (*reuse step*): the proposed solution $\mathtt{y}^{\mathtt{tgt}}$ is either $\mathtt{y}^s$ (reused as such) or modified to take into account the mismatch between $\mathtt{x}^s$ and $\mathtt{x}^{\mathtt{tgt}}$. (3) The pair $(\mathtt{x}^{\mathtt{tgt}}, \mathtt{y}^{\mathtt{tgt}})$ is then tested to see whether $\mathtt{y}^{\mathtt{tgt}}$ correctly solves $\mathtt{x}^{\mathtt{tgt}}$ and, if not, $\mathtt{y}^{\mathtt{tgt}}$ is repaired for this purpose (*revise step*); this step is often made by a human. (4) Finally, the newly formed case $(\mathtt{x}^{\mathtt{tgt}}, \mathtt{y}^{\mathtt{tgt}})$ is added to $\mathtt{CB}$ if this storage is judged appropriate (*retain step*). In some applications, the retrieve step returns several source cases that the reuse step combines.

Consider an example related to a case-based system in the cooking domain [10]. This system lets users formulate queries to retrieve cooking recipes. It provides adapted recipes when the execution of a query does not give any result. For instance, a user may want to find a recipe for an *apple pie with chocolate*. There is no such recipe in the case base but there exists a recipe of a *chocolate pear pie* which is the best match to the initial query (*retrieve step*). The *reuse step* may consist in adapting the recipe by replacing pears with apples, with an adjustment of the ingredient quantities This adapted recipe is proposed to a user who may give a feedback useful to improve this recipe. (*revise step*). Finally, the newly formed case is added to the case base (*retain step*).

2.2 RDF(S): a Semantic Web Technology

The term RDF(S) refers to the combined use of RDF, RDF Schema and SPARQL technologies.

RDF. Resource Description Framework [12] (RDF) provides a way for representing data, by using a metadata model based on labeled directed graphs. Information is represented using statements, called triples, of the form $\langle subject \ \ predicate \ \ object \rangle$. The subject s is a resource, the predicate p is a property, and the object o is either a resource or a literal value.

RDFS. RDF Schema [5] (RDFS) adds a logic upon the RDF model. A new set of specific properties are introduced: `rdfs:subclassof` (resp. `rdfs:subpropertyof`) allows to create a hierarchy between classes (resp. properties). `rdfs:domain` (resp. `rdfs:range`) applies for a property and adds a constraint about the type of the resource which is in *subject* (resp. *object*) position for a triple. In the remainder of the article, short names will be used for these properties: `a` for `rdf:type`, `domain` for `rdfs:domain` and `range` for `rdfs:range`. The inference relation $\vdash$ is based on a set of inference rules [5].

SPARQL. SPARQL is the language recommended by the World Wide Web Consortium (W3C) to query RDF data [16]. Consider the following informal query:

$$\mathcal{Q} = \left| \begin{array}{l} \text{“Give me the letters sent by Henri Poincaré to} \\ \text{ mathematicians between 1885 and 1890”} \end{array} \right.$$

This query can be represented using SPARQL:

$$\mathtt{Q} = \left| \begin{array}{l} \texttt{SELECT ?l} \\ \texttt{WHERE \{?l a letter .} \\ \texttt{\quad\quad ?l sentBy henriPoincaré .} \\ \texttt{\quad\quad ?l sentTo ?person .} \\ \texttt{\quad\quad ?person a Mathematician .} \\ \texttt{\quad\quad ?l sentInYear ?y .} \\ \texttt{\quad\quad FILTER(?y >= 1885 AND ?y <= 1890)\}} \end{array} \right.$$

For the sake of simplicity, in the remainder of the article, queries are presented in an informal way though all of them correspond to actual SPARQL queries.

3 The Henri Poincaré Correspondence: Edition and Online Publishing

3.1 Context and Works

Through the Henri Poincaré website, anyone can access the letters of the Henri Poincaré correspondence. About 60% of the letters are associated with a plain text transcription (in *XML* or LaTeX), a critical apparatus and a set of metadata. Metadata can either refer to the letter as a physical object (writing date, place of expedition, sender, recipient, etc.) or to the content of the letter (scientific topics discussed, people quoted, etc.). The content management system Omeka S [4] has been used to create this website and to publish data related

to this correspondence. This platform enables the web-publishing and sharing of cultural collections from institutions (museums, archives, etc.). It operates with a MySQL database. A search engine using the Solr tool, which allows plain text search, has been implemented to retrieve transcribed letters. Although appropriate to some situations, this search engine suffers from a lack of expressiveness. In practice, historians often need to express more complex and structured queries. As an example, one can be interested in finding the letters sent by Henri Poincaré to members of his family in which he mentioned his classmates at the time he was a student of the "École polytechnique". To address this issue, an RDFS base has been initialized and is daily updated by translating the Omeka S content to Turtle files.[2] This database is structured using the *Archives Henri-Poincaré Ontology* which, in particular, describes resources and relations in the context of scientific correspondences. It gathers classes and properties related to persons, institutions, places, documents (e.g. books, articles, letters, etc.). This ontology is aligned with the use of several standard vocabularies (dcterms, bibo, rel, etc.). Three SPARQL querying modes have been created and are available on the website [7]. The *classical mode* requires to directly write SPARQL queries. The *form-based mode* proposes a set of input fields to assist the user in the generation of the query. The *graphical mode* presents a graph-based interface which lets the user manipulate nodes and edges to formulate the query.

As described in the introduction of this article, the annotation process is a tedious work which justifies the need of a dedicated editor to assist the user. This system should enable an efficient interactive update of an RDFS base, by visualizing the already edited statements and providing suggestions appropriate to the current annotation context.

3.2 Proposal of an Annotation Tool

A suggestion system has been developed to assist the user during the annotation process. This system has been implemented in Java. It comes with a set of parameters in order to connect to any given RDF base. It can be a file system base or a base reachable through a SPARQL endpoint. Different engines can be used to dialog with the RDF base (e.g. Jena [13], Corese [9]).

Associated to this system, a web user interface has been developed to use and compare the different versions of the system. This interface proposes an autocomplete mechanism that uses the suggestion system for providing values. The interface is common to all versions of the system. The tool enables the visualization and update of RDF databases. Three fields are available to set the values of subject, predicate and object. The use of prefixes has been implemented to improve the readability of the tool. The list of existing namespaces and associated prefixes is accessible through the "Prefixe" tab. When editing a triple, the editor displays the associated *context*. This corresponds to the set of already edited triples related to the current subject resource. For example, if the current subject is `letter11`, the editor displays the results of the execution of

[2] Turtle is a RDF serialization which is easily readable [8].

the query SELECT `?p ?o` WHERE {`letter11 ?p ?o`}. This context is refreshed each time the value of the subject field is updated. When a new triple is created and inserted into the database, it is added to the current context. An excerpt of this interface is presented in Fig. 1. The full interface associated with several use cases is the subject of a presentation video accessible online.[3] The first two versions of the suggestion engine are presented here.

The *basic editor* assists the user by proposing an autocomplete mechanism in which the suggestions are ranked using the alphabetical order. The proposed suggestions do not depend neither on the current annotation problem nor on the available data and knowledge.

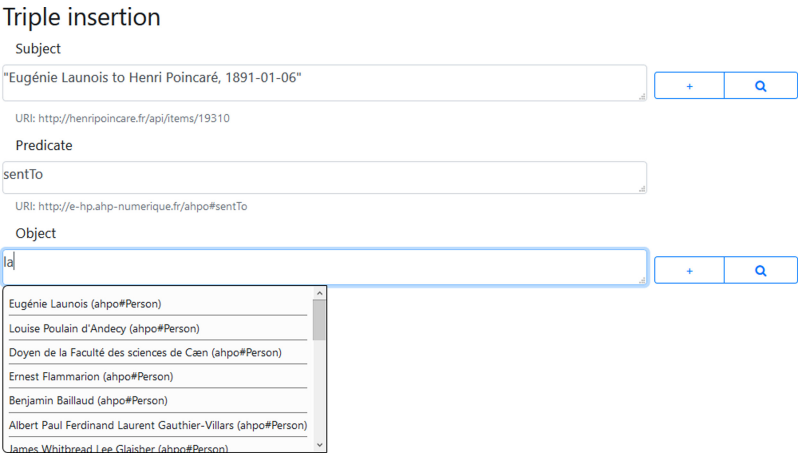

Fig. 1. An excerpt of the RDFWebEditor4Humanities interface.

The *deductive editor* benefits from the use of RDFS knowledge for ranking the suggestions provided to the user. The notion of *annotation question* is introduced: this corresponds to a triple for which 1, 2 or the 3 fields are unknown, and for which a field is currently being edited. This field is represented by using a frame around an existential variable (i.e. `?p`, `?o`). For instance, ⟨*s* `?p` `?o`⟩ corresponds to an annotation question type for which the subject is known, the predicate is currently being edited and the object is unknown. There exist twelve different annotation question types. For each of them, the knowledge about the `domain` and `range` can be used to rank the potential values for the targeted field.

Consider the annotation question ⟨`letter11 sentBy` `?o`⟩ which is of the type ⟨*s* *p* `?o`⟩. The objective here is to provide appropriate suggestions by listing and ranking the potential values for the object field. For this version of the

[3] https://videos.ahp-numerique.fr/videos/watch/0d544e5b-b4be-423e-9497-216f29ab44f3.

editor, the top suggestions are resources of the classes `Person` and `Institution`. In the ontology, `Person` and `Institution` are `range` of the property `sentBy`. This knowledge is used to favor the instances of these classes. However, the resources that do not explicitly belong to these classes are still suggested because RDFS works with "open world assumption".[4]

There exist different rules which can be used to retrieve a list of potential values and whose applications depend on the type of the annotation question. For instance, the rule used to answer the annotation question presented in the example below is called `rangePred` and may apply for the annotation questions of the type $\langle s \;\; p \;\; \boxed{\texttt{?o}} \rangle$ and $\langle \texttt{?s} \;\; p \;\; \boxed{\texttt{?o}} \rangle$. To answer an annotation question, a count is computed for each potential value `?v` based on the number of rule applications which retrieved this value. The final list of suggestions is ranked according to this count (in decreasing order). For the potential values with the same count, the alphabetical order is used. 6 different rules have been defined. `domainPred` uses the knowledge about the domain of a given property p and may apply for the annotation questions of the type $\langle \boxed{\texttt{?s}} \;\; p \;\; o \rangle$ and $\langle \boxed{\texttt{?s}} \;\; p \;\; \texttt{?o} \rangle$. The application of `predProperty` increases the count of each candidate solution which is defined as an `rdf:Property` for the annotation questions in which the target is the value in predicate position. `subjectInDomain` uses the value s defined in subject position: if s is an instance of a class `D`, each candidate value having `D` as domain will have its count incremented. It may apply for the annotation questions of the type $\langle s \;\; \boxed{\texttt{?p}} \;\; \texttt{?o} \rangle$ and $\langle s \;\; \boxed{\texttt{?p}} \;\; o \rangle$. In a symmetrical way, `objectInRange` uses the range of the value o and may apply for the annotation questions of the type $\langle \texttt{?s} \;\; \boxed{\texttt{?p}} \;\; o \rangle$ and $\langle s \;\; \boxed{\texttt{?p}} \;\; o \rangle$. For the annotation questions of the type $\langle \boxed{\texttt{?s}} \;\; \texttt{?p} \;\; o \rangle$, each candidate value which is in the domain of a property whose range contains o will have its count incremented. This rule is called `subjImRel` and may apply in a symmetrical way for the annotation questions of the type $\langle s \;\; \texttt{?p} \;\; \boxed{\texttt{?o}} \rangle$ by using the value s.

4 A Case-Based Editor

4.1 Case-Based Content Annotation

The use of RDFS deduction (as described in Sect. 3.2) brings a first improvement to the suggestion system by using the knowledge about the `domain` and `range` of the properties defined in the base. However, in some situations, this is not enough to propose the most appropriate resources for the current editing question. As an example, consider a triple currently being edited for which the subject is an instance of `Letter` (`letter2100`), the predicate is `sentTo` and for which suggestions for the object field are expected. As the class `Person` is `range` of

[4] If a fact is not asserted, it does not imply that this fact is false. In this situation, there may exist a resource r that is intended to represent a person (resp. institution) but is such that the triple $\langle r \;\; \texttt{a} \;\; \texttt{Person} \rangle$ (resp. $\langle r \;\; \texttt{a} \;\; \texttt{Institution} \rangle$) cannot be entailed by the *current* RDFS base. Therefore, r can be suggested as well, though further in the suggestion list.

the property `sentTo`, the system will favor the instances of this class in the suggestion list. But the problem is that there are many instances of this class in the base,[5] and there is no guarantee that the appropriate value will be among the first suggestions in the list. Indeed, for the values with the same count, the alphabetical order is used.

An alternative way to obtain a relevant ranking of the suggestion list would be to follow the CBR methodology: in the current situation, pieces of information from similar situations can be reused. In this framework, an annotation problem $\mathtt{x}^{\mathtt{tgt}}$ is composed of an editing question and a context. For editing questions of the type $\langle s \;\; p \;\; \boxed{\mathtt{?o}} \rangle$, it is defined as follows:

$$\mathtt{x}^{\mathtt{tgt}} = \boxed{\begin{array}{l}\textbf{question:} \langle subj^{\mathtt{tgt}} \;\; pred^{\mathtt{tgt}} \;\; \boxed{\mathtt{?o}} \rangle \\ \textbf{context:} \text{ the set of triples related to } subj^{\mathtt{tgt}}\end{array}}$$

For the running example, this gives:

$$\mathtt{x}^{\mathtt{tgt}} = \boxed{\begin{array}{ll}\textbf{question:} & \langle \mathtt{letter2100} \;\; \mathtt{sentTo} \;\; \boxed{\mathtt{?o}} \rangle \\ \textbf{context:} & \langle \mathtt{letter2100} \;\; \mathtt{sentBy} \;\; \mathtt{henriPoincaré} \rangle \\ & \langle \mathtt{letter2100} \;\; \mathtt{hasTopic} \;\; \mathtt{écolePolytechnique} \rangle \\ & \langle \mathtt{letter2100} \;\; \mathtt{quotes} \;\; \mathtt{paulAppell} \rangle\end{array}}$$

The case base is the RDF database $\mathcal{D}_{\text{HP}}$. A source case is given by a triple $\langle subj^{\mathtt{s}} \;\; pred^{\mathtt{s}} \;\; obj^{\mathtt{s}} \rangle$ of $\mathcal{D}_{\text{HP}}$, considered among all the triples of $\mathcal{D}_{\text{HP}}$, and which, in relation to $\mathtt{x}^{\mathtt{tgt}}$, can be decomposed into a problem $\mathtt{x}^s$ and a solution $\mathtt{y}^s$:

$$\mathtt{x}^s = \boxed{\begin{array}{l}\textbf{question:} \langle subj^{\mathtt{s}} \;\; pred^{\mathtt{s}} \;\; \boxed{\mathtt{?o}} \rangle \\ \textbf{context:} \text{ the database } \mathcal{D}_{\text{HP}}\end{array}}$$

and the solution $\mathtt{y}^s = obj^{\mathtt{s}}$. For the purpose of this example, consider an excerpt $\mathcal{D}_{ex}$ of the Henri Poincaré correspondence database $\mathcal{D}_{\text{HP}}$ composed of the letters related to the following instances of the class `Person`: `göstaMittagLeffler`, `alineBoutroux`, `eugénieLaunois`, `felixKlein` and `henriPoincaré`. How should the list composed of these 5 resources be ordered? To propose a solution to this problem, the method consists in retrieving the cases which correspond the best to the current annotation problem. At each source case $\mathtt{x}^s$ is associated a value $\mathtt{y}^s$ which is used as a candidate solution to $\mathtt{x}^{\mathtt{tgt}}$. A count is computed for ranking the candidate solutions. This corresponds to the number of letters having this value associated with the property `sentTo`. An initial SPARQL query $\mathcal{Q}$ based on $\mathtt{x}^{\mathtt{tgt}}$ is defined to compute this count. For the running example, it gives:

$$\mathcal{Q} = \left| \begin{array}{l}\text{“Give me, for each potential value } \mathtt{?o}\text{, the number of letters} \\ \quad \text{having this value associated with the } \mathtt{sentTo} \text{ property} \\ \quad \text{where letters have } \mathtt{écolePolytechnique} \text{ as topic,} \\ \quad \text{quote } \mathtt{paulAppell} \text{ and have been sent by } \mathtt{henriPoincaré}\text{”}\end{array} \right.$$

[5] At the time of writing this article, there are around 1800 persons defined within the database.

The execution of $\mathcal{Q}$ on $\mathcal{D}_{ex}$ returns an empty set of results. Indeed, it is uncommon to find two different letters having exactly the same context. So, the question is to find a method to retrieve the most similar source cases. This issue can be addressed by using SPARQL query transformations. An engine has already been designed to manage transformation rules and has proven useful in different contexts including the search in the Henri Poincaré correspondence corpus and the case-based cooking system Taaable [6]. Rules are configurated by the user and can be general or context-dependent. To each rule is associated a cost, corresponding to a query transformation cost. Two rules are considered in the running example:

- $r_{\texttt{exchange}}$: exchanges the sender and recipient of the letter (cost of 2);
- $r_{\texttt{genObjInst}}$: generalizes a class instance in object position (cost of 3).

A search tree can be explored starting from the initial query $\mathcal{Q}$ by applying one or several successive transformation rules. A maximum cost is defined to limit the depth of the search tree exploration. For this application, this maximum cost is set to 10.

At depth 1, the application of the rule $r_{\texttt{exchange}}$ on $\mathcal{Q}$ generates the query $\mathcal{Q}_1$ with a cost of 2 (the modified part of the query is underlined):

$\mathcal{Q}_1 =$ "Give me, for each potential value `?o`, the number of letters having this value associated with the `sentBy` property where letters have `écolePolytechnique` as topic, quote `paulAppell` and have been received by `henriPoincaré`"

The result of the execution of $\mathcal{Q}_1$ on $\mathcal{D}_{ex}$ is: [{`eugénieLaunois` : 2}, {`alineBoutroux` : 1}]. Three applications of the rule $r_{\texttt{genObjInst}}$ exist at depth 1, each of them for a cost of 3. The first one applies for the people quoted, by replacing `paulAppell` by any instance of the class `Mathematician` (because Paul Appell belongs to that class), the second one applies for the sender of the letter, and the last one applies for `écolePolytechnique` by replacing the value by any instance of the class `Topic`. The generated queries are $\mathcal{Q}_2$, $\mathcal{Q}_3$ and $\mathcal{Q}_4$:

$\mathcal{Q}_2 =$ "Give me, for each potential value `?o`, the number of letters having this value associated with the `sentTo` property where letters have `écolePolytechnique` as topic, quote a `Mathematician` and have been sent by `henriPoincaré`"

$\mathcal{Q}_3 =$ Give me, for each potential value `?o`, the number of letters having this value associated with the `sentTo` property where letters have `écolePolytechnique` as topic, quote `paulAppell` and have been sent by a `Mathematician`

$\mathcal{Q}_4 =$ Give me, for each potential value `?o`, the number of letters having this value associated with the `sentTo` property where letters have a defined topic, quote `paulAppell` and have been sent by `henriPoincaré`

The execution of $\mathcal{Q}_2$ on $\mathcal{D}_{ex}$ gives: [{`eugénieLaunois` : 138}, {`alineBoutroux` : 4}]. The executions of $\mathcal{Q}_3$ and $\mathcal{Q}_4$ give no result. As the maximum cost has been set to 10, it is possible to continue the tree exploration on the different branches to find new possible suggestions. At depth 2, the application of $r_{\texttt{genObjInst}}$ on $\mathcal{Q}_2$ (applied for the topic) generates the query:

$$\mathcal{Q}_{21} = \left| \begin{array}{l} \text{Give me, for each potential value } \texttt{?o}\text{, the number of letters} \\ \quad \text{having this value associated with the } \texttt{sentTo} \text{ property} \\ \quad \text{where letters have a defined topic,} \\ \quad \text{quote a } \texttt{Mathematician} \text{ and have been sent by } \texttt{henriPoincaré} \end{array} \right.$$

The execution of $\mathcal{Q}_{21}$ on $\mathcal{D}_{ex}$ gives: [{`eugénieLaunois` : 280}, {`göstaMittagLeffler` : 74}, {`alineBoutroux` : 17}]. At depth 3, the application of $r_{\texttt{genObjInst}}$ on $\mathcal{Q}_{21}$ (applied for the sender of the letter) generates the query:

$$\mathcal{Q}_{211} = \left| \begin{array}{l} \text{Give me, for each potential value } \texttt{?o}\text{, the number of letters} \\ \quad \text{having this value associated with the } \texttt{sentTo} \text{ property} \\ \quad \text{where letters have a defined topic,} \\ \quad \text{quote a } \texttt{Mathematician} \text{ and have been sent by a } \texttt{Mathematician} \end{array} \right.$$

The execution of $\mathcal{Q}_{211}$ on $\mathcal{D}_{ex}$ gives: [{`eugénieLaunois` : 305}, {`henriPoincaré` : 219}, {`göstaMittagLeffler` : 141}, {`felixKlein` : 25}, {`alineBoutroux` : 21}]. The other possible rule applications (considering maximum cost) generate queries already generated by other combinations or which give the same resources but with a greater cost.

The final list of suggestions is ranked by ordering the resources based on the required minimal transformation cost. For resources with the same minimal cost, the count related to the execution of the query associated to this cost is used (in a decreasing order). For the running example, this gives, for the first 5 suggestions from number 1 to number 5: `eugénieLaunois`, `alineBoutroux`, `göstaMittagLeffler`, `henriPoincaré`[6] and `felixKlein`. The remainder of the suggestions is composed of all the resources of the database ranked using the alphabetical order.

This approach constitutes the *retrieve* step of the CBR model. The *reuse* step is a *reuse as such* approach: there is no modification of the proposed resources. After this, the user chooses the appropriate resource, which could be considered as a *revise* step. Then the edited triple is inserted into the database (*retain* step).

4.2 Combining RDFS Deduction with CBR

The last version of the editor combines the use of RDFS deduction with CBR. It takes advantage of both the knowledge about the resources similar to the

[6] This suggestion could be removed if the system knows that the recipient of a letter cannot be its sender. This is considered again in the future work part of this article conclusion.

one being edited and the `domain` and `range` of the properties used during the editing. The resources found using CBR are on top of the suggestion list. For the other resources, a count is computed for ranking the potential values as presented in Sect. 3.2. Consider the example presented above, in which the editing question was ⟨`letter2100 sentTo ?o`⟩ and suggestions for the object field were expected. Using the CBR version of the system, the first five suggestions are resources which seem to be pertinent considering the current editing context and by looking for the similar objects in the database. But for the remainder of the suggestions, only the alphabetical order is used for ranking. This can be addressed by using the `range` of the property `sentTo` (as explained in Sect. 3.2) for ranking the second part of the suggestions list (from the 6^{th} value). As `Person` is `range` of the property `sentBy`, all the instances of this class would be higher in the suggestions list than instances of other classes.

5 Evaluation

The goal of the evaluation is to compare the efficiency of the different versions of the system for concrete annotation situations. The first evaluation is human-based through a user who will test and compare the four versions. A second evaluation is managed through a dedicated program and will provide objective measures. Both evaluations focus on a subset of 7 properties among the most frequently used when editing letters: `sentBy` defines the sender; `sentTo` defines the recipient; `hasTopic` gives one of the topics; `archivedAt` specifies the place of archive; `hasReply` gives a letter responding to the current letter; `replies` gives a letter to which the current letter responds; `citeName` refers to a person mentioned in the letter transcription.

5.1 Human Evaluation

This evaluation involves a single user who is one of the people in charge of the editing of the Henri Poincaré correspondence corpus. He was using Omeka S before moving on to the system presented in this article. He had no previous experience with this tool at the time he carried out the evaluation. The test set is composed of 10 letters which have been randomly chosen from a set of 30 unpublished letters from the Henri Poincaré correspondence corpus. This set constitutes a real annotation case with respect to the already edited letters in the corpus database. The new items from the evaluation corpus have been edited using Omeka S before the start of the evaluation, so as to ensure that no version of the system would suffer from being the first one to be evaluated. For each version (presented in a random order), the user edits (i.e. create the triples) the same 10 letters using the interface provided with the tool. Before switching to the next version, the RDF database is reset to correspond to the initial state.

After having edited the complete set of letters with one version, the user is invited to complete a survey and to provide feedback about this version. This survey insists on the appreciation of the autocomplete mechanism efficiency (but

Table 1. The average score (on a 1 to 7 scale) associated with the suggestions provided by the four versions of the system.

	Basic editor	Deductive editor	Cased-based editor	Combination editor
Average score	3.4	5.7	5.3	7

experience feedback about the user interface is also expected). For each property, the user is invited to attribute a score by using a *Likert scale* [2], from 1 (not at all relevant) to 7 (very relevant) to characterize the relevance of the suggestions provided for annotation questions linked to that property.

What emerges of this evaluation is that the combination editor has been perceived as the most efficient, and this for all the properties of the evaluation. The average scores of all property evaluations are given on Table 1. The basic editor is the version that obtained the lowest score. It has been perceived as "not assisting the annotation", but still not causing any problems to the user. The deductive and case-based editors got high average scores. However, in situations in which the retrieval of source cases leads to an empty set of cases, the CBR engine only uses the alphabetical order for ranking the list of suggestions, and may provide irrelevant resources. This caused frustration for the user and explains why the average score of the CBR engine is lower than the deductive engine. Combining the two engines is a good method to counter these situations.

Furthermore, the interface associated with the tool helped avoiding the mistakes described in the introduction: it prevents the insertion of triples which already exist in the database (*duplication mistake*), the type of the selected resource is always visible (*ambiguity mistake*), and the use of labels simplify the management of resources for the user (*typing mistake*). The user felt in control when he was performing actions to alter the database. At the end of the evaluation, the user has proceed with a few more tests to compare Omeka S with the last version of the editor. He has estimated that the time required for the annotation of a letter using the combination editor was about half the time he needed with Omeka S.

5.2 Automatic Evaluation

The aim of the automatic evaluation is to compare the performances of the different versions of the tool by computing measures. The chosen measures are related to the rank of the expected value $\texttt{rank}(aq)$ where aq is the current annotation question. $\texttt{rank}(aq) = 1$ means that the associated value is the first in the suggestion list. In others words, the lower the rank is, the better the version of the system is.

The RDF graph of the Henri Poincaré correspondence $\mathcal{G}_{\text{HP}}$ has been used as a test set. This graph is formed by the union of the database $\mathcal{D}_{\text{HP}}$ and the ontology $\mathcal{O}_{\text{HP}}$: $\mathcal{G}_{\text{HP}} = \mathcal{D}_{\text{HP}} \cup \mathcal{O}_{\text{HP}}$. At the time of writing this article, the RDF database $\mathcal{D}_{\text{HP}}$ is composed of around 220 000 triples. The database and ontology triples are stored in Turtle files. For this evaluation, the application of the inference

Table 2. Rank measures for the suggestions provided by the four versions of the system.

	Basic editor	Deductive editor	Case-based editor	Combination editor
$rank \leq 15$	11.3%	22.11%	49.0%	49.5%
$rank \leq 10$	7.1%	21.15%	43.2%	43.2%
$rank \leq 5$	2.7%	19.23%	33.6%	34.7%

rules mentioned in Sect. 2 has been considered. Different classes of items exist in the database (e.g. `Letter`, `Person`, `Article`, etc.) but, for this evaluation, the focus is put on the editing of letters. A set of 100 letters is randomly extracted from the existing set of annotated letters. For each letter of this set, the related triples part of the context are used to simulate annotation questions for which the answer is already known. For each triple, the order of editing of the 3 fields (subject, predicate and object) is considered in a random order so as to include various annotation question types in the evaluation. For each annotation question aq, the four suggestion engines are called to provide an ordered suggestion list. The rank of the expected value $\texttt{rank}(aq)$ in the list is saved for each version and is added to the related multi-set $\texttt{Ranks}(system)$. At the end of the evaluation, measures related to the elements of $\texttt{Ranks}(system)$ are computed. These measures correspond to the percentage of annotation questions for which the expected value was given among the n first propositions ($rank \leq n$).

The results of this evaluation are given in Table 2 for each version of the system. Different values of n have been chosen (5, 10 and 15) but the evolution of the efficiency of the versions is the same in all situations. This shows that the combination editor provides the best results for the different annotation questions related to this evaluation because it suggests the appropriate value more often. It is thus more likely to assist the user during the annotation process.

Although not used as a measure during the evaluation, the computing time has been considered. It corresponds to the time needed to provide the suggestion list for an annotation question. Indeed, the reaction time to requests should be considered in a human interaction system especially since this system is using an autocomplete mechanism for which a user expects no latency. The computation time is more important when using the combination editor but this stays low enough not to impact the user (around 1 s for a standard laptop).

6 Discussion and Related Work

The method presented in Sect. 4 is inspired from the UTILIS system [11]. This system introduces the idea of looking for resources similar to the one being edited to suggest values that might be appropriate to the current annotation problem. But the form of query relaxation proposed is different as it mainly uses generalization rules. The engine presented in this article allows the users to define their own rules to correspond to a specific database. Combined with the

use of RDFS knowledge, it allows to propose suggestions appropriate to various annotation question types.

One of the most frequently used tools for editing Semantic Web data is Protégé [14]. When editing an instance of a specific class, Protégé uses the `domain` and `range` of the properties to make suggestions for predicate and object values. But these suggestions do not have profit from the already edited triples. Other approaches exist to assist the editing of RDF databases, several of them being based on natural language processing. The GINO editing tool [3] proposes the use of a guided and controlled natural language which lets the user specify sentences corresponding to statements. The main idea is that the principles of Semantic Web are sometimes not easily apprehended by non specialists, and thus should be encapsulated within a more user-friendly system. The syntax of this language is close to English syntax (e.g. "There is a mount named Everest", "The height of mount Everest is 29 029 feet", etc.). A suggestion mechanism proposes classes, instances and properties to complete the current annotation. These suggestions are ranked alphabetically and are consistent with the defined ontology. The main challenge in this system is the interpretation of the user request to build triples from sentences.

More generally, the tool that is presented in this article could be categorized as a recommender system. These systems intend to assist the user by presenting information likely to interest her/him. Different recommender systems, such as the one presented here, use case-based recommendation [18]. A great variety of methods exists, and the tool presented in this article could benefit from several of them. As an example, the involvement of the user in the suggestion proposal mechanism is considered. The explainability of the tool could be reinforced as it may be important to understand why some resources are more favored than others. This system could also benefit from the use of a preference-based feedback system which could improve the results of the tool in several situations, make the user feel included and thus reinforce the positive view about the tool. On the other hand, the query transformation mechanism which has been used in this application framework could be reused in other recommender systems.

7 Conclusion

The use of Semantic Web technologies has proven useful for the corpus of the correspondence of Henri Poincaré. A manual annotation process has been chosen to edit data related to items of this corpus (e.g. *letters*, *persons*, *places*, etc.) This process has been identified tedious for users in charge of the editing. To deal with this issue, a tool providing a suggestion system has been proposed. It intends to be a general tool for the editing of Semantic Web data. It uses some inferences with RDFS entailment combined with a CBR methodology. Different versions of the system have been implemented. The first version ranks the potential values by using the alphabetical order. The second version takes advantage of the knowledge about the `domain` and `range` for the properties of the base. The third version uses CBR to exploit the knowledge about similar edited resources. The

last version is a combination of the two latter versions. Two different evaluations have been conducted. The human evaluation allowed to compare the different versions of the system between them and with the current existing annotation system (Omeka S). The automatic evaluation brought metrics by comparing, for a selected set of annotation questions, the suggestions of the different versions of the system. The relevance of the suggestions and the computing time have been taken into account. As explained in Sect. 5, while the metrics computed by the automatic evaluation show that the use of CBR alone brought better results than the use of RDFS deduction alone, the case-based version is sometimes insufficient and can provide irrelevant resources in some situations. The hypothesis 1 is validated by the automatic evaluation but not by the human evaluation. For both evaluations, the results show that the last version of the system combining the use of RDFS deduction with CBR is the most efficient and thus validates the hypothesis 2 stated in the introduction. However, in some situations, this system tends to show some limitations. For instance, consider the annotation question presented in Sect. 4. Both third and fourth versions of the system proposes `henriPoincaré` as a plausible answer although he is already defined as the sender of the current edited letter. A way to deal with this issue would be the use of some domain knowledge used as integrity constraints. For this example, the piece of knowledge "A resource cannot be at the same time the recipient and the sender of letter" could be used to prevent the suggestion of `henriPoincaré` as the recipient of a letter he sent. Another point observed during both human and automatic evaluations is that the order of editing of the different properties affects greatly the efficiency of the suggestion engine. Indeed, some properties values give more information about the resource than others, and thus having these values filled first should improve the ranking of the suggestions. Main challenge is to find the best order of editing for the properties of the base. This constitutes a future work. Another future work is related to the use of a more expressive logic than RDFS such as OWL-DL. A logic containing a form of negation would enable to remove some values from the list of potential values. However, such an extension could affect the computation time and its implementation should be investigated.

Although the RDFWebEditor4Humanities tool is now used for the editing of RDF data, Omeka S still provides some useful functionalities. It forms a stable environment for both editing and publishing of the items related to that corpus. Two solutions are considered: the first one consists in integrating some functionalities of Omeka S in the new annotation tool; the second one considers the creation of a new Omeka S module which would call the suggestion system to assist the user during the annotation process.

Acknowledgments. This work was supported partly by the french PIA project "Lorraine Université d'Excellence", reference ANR-15-IDEX-04-LUE. It was also supported by the CPER LCHN (Contrat de Plan État-Région Lorrain "Langues, Connaissances et Humanités Numériques") that financed engineer Ismaël Bada who participated to this project. We greatly thank Mickaël Smodis who is a final user of the tool and

who participated to the human evaluation and Laurent Rollet who provided us with unpublished letters of the Henri Poincaré correspondence.

References

1. Aamodt, A., Plaza, E.: Case-based reasoning: foundational issues, methodological variations, and system approaches. AI Commun. **7**(1), 39–59 (1994)
2. Allen, I.E., Seaman, C.A.: Likert scales and data analyses. Qual. Prog. **40**(7), 64–65 (2007)
3. Bernstein, A., Kaufmann, E.: GINO – a guided input natural language ontology editor. In: Cruz, I., et al. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 144–157. Springer, Heidelberg (2006). https://doi.org/10.1007/11926078_11
4. Boulaire, C., Carabelli, R.: Du digital naive au bricoleur numérique: les images et le logiciel Omeka. In: Cavalié, É., Clavert, F., Legendre, O., Martin, D. (eds.) Expérimenter les humanités numériques. Des outils individuels aux projets collectifs. Les Presses de l'Université de Montréal, Montréal, Québec, pp. 81–103 (2017)
5. Brickley, D., Guha, R.V.: RDF Schema 1.1, W3C recommendation (2014). https://www.w3.org/TR/rdf-schema/. Accessed Aug 2020
6. Bruneau, O., Gaillard, E., Lasolle, N., Lieber, J., Nauer, E., Reynaud, J.: A SPARQL query transformation rule language—application to retrieval and adaptation in case-based reasoning. In: Aha, D.W., Lieber, J. (eds.) ICCBR 2017. LNCS (LNAI), vol. 10339, pp. 76–91. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61030-6_6
7. Bruneau, O., Lasolle, N., Lieber, J., Nauer, E., Pavlova, S., Rollet, L.: Applying and developing semantic web technologies for exploiting a corpus in history of science: the case study of the Henri Poincaré correspondence. Semant. Web. IOS Press Journal (2020)
8. Carothers, G., Prud'hommeaux, E.: RDF 1.1 Turtle (2014). http://www.w3.org/TR/2014/REC-turtle-20140225/. Accessed Aug 2020
9. Corby, O., Dieng-Kuntz, R., Faron Zucker, C.: Querying the semantic web with corese search engine. In: López de Mántaras, R., Saitta, L. (eds.) European Conference on Artificial Intelligence, Valence, Spain, pp. 705–709, August 2004
10. Cordier, A., et al.: Taaable: a case-based system for personalized cooking. In: Montani, S., Jain, L.C. (eds.) Successful Case-Based Reasoning Applications-2. SCI, vol. 494, pp. 121–162. Springer, Cham (2014). https://doi.org/10.1007/978-3-642-38736-4_7
11. Hermann, A., Ferré, S., Ducassé, M.: An interactive guidance process supporting consistent updates of RDFS graphs. In: ten Teije, A., et al. (eds.) EKAW 2012. LNCS (LNAI), vol. 7603, pp. 185–199. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33876-2_18
12. Manola, F., Miller, E., McBride, B., et al.: RDF Primer (2004). https://www.w3.org/TR/rdf-primer. Accessed Aug 2020
13. McBride, B.: Jena: a Semantic Web toolkit. IEEE Internet Comput. **6**(6), 55–59 (2002)
14. Noy, N.F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R.W., Musen, M.A.: Creating semantic web contents with Protégé-2000. IEEE Intell. Syst. **16**(2), 60–71 (2001)
15. Poincaré, H.: La Science et l'Hypothèse. Flammarion, Paris (1902)
16. Prud'hommeaux, E.: SPARQL Query Language for RDF, W3C Recommendation (2008). http://www.w3.org/TR/rdf-sparql-query/. Accessed Aug 2020

17. Riesbeck, C.K., Schank, R.C.: Inside Case-Based Reasoning. Lawrence Erlbaum Associates Inc., Hillsdale (1989)
18. Smyth, B.: Case-based recommendation. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) The Adaptive Web. LNCS, vol. 4321, pp. 342–376. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72079-9_11