Navigation-induced Knowledge Engineering by Example

A New Paradigm For Knowledge Engineering by the Masses

Sebastian Hellmann[1], Jens Lehmann[1], Jörg Unbehauen[1], Claus Stadler[1], Thanh Nghia Lam[1], Markus Strohmaier[2]

[1] Department of Computer Science, University of Leipzig
Johannisgasse 26, 04103 Leipzig
hellmann|lehmann|unbehauen|cstadler@informatik.uni-leipzig.de,
[2] Graz University of Technology and Know-Center
Inffeldgasse 21a, 8010 Graz, Austria
markus.strohmaier@tugraz.at

Abstract. Knowledge Engineering is a costly, tedious and often time-consuming task, for which light-weight processes are desperately needed. In this paper, we present a new paradigm - Navigation-induced Knowledge Engineering by Example (NKE) - to address this problem by producing structured knowledge as a result of users navigating through an information system. Thereby, NKE aims to reduce the costs associated with knowledge engineering by framing it as navigation. We introduce and define the NKE paradigm and demonstrate it with a proof-of-concept prototype which creates OWL class expressions based on users navigating in a collection of resources. The overall contribution of this paper is twofold: (i) it introduces a novel paradigm for knowledge engineering and (ii) it provides evidence for its technical feasibility.

Keywords: Navigation, Knowledge Engineering, Paradigm, Methodology, Ontology Learning, Search, OWL

1 Introduction

Over the past years, structured data has increasingly become available on the World Wide Web (WWW). Yet, the actual usage of this data still poses significant barriers for lay users. One of the main drawbacks to the utilization of the structured data on the WWW lies in the blatant cognitive gap between the informational needs of users and the structure of existing knowledge bases. In this paper, we propose a novel paradigm - Navigation-induced Knowledge Engineering by Example (NKE) - which aims to bridge this gap.

Due to the sheer size of large knowledge bases, users can hardly know which identifiers are available or useful for the construction of axioms or queries. As a consequence, users might not be able to express their informational need in a

structured form. Yet, users often have a very precise idea of what kind of results they would like to retrieve. A historian, for example, searching DBpedia [16] for `ancient Greek law philosophers influenced by Plato` can easily name some examples and - when presented with a selection of prospective results - she will be able to quickly identify correct and incorrect results. However, she might not be able to efficiently construct a formal query adhering to the large DBpedia knowledge base.

In this paper, we will argue and show that Navigation-induced Knowledge Engineering by Example can tackle important parts of the above described information gap problem.

In NKE, the informational need of a user is approximated, e.g. by letting the user formulate preferences or simply by browsing an application. From this interaction, so called *positive and negative examples* can be inferred that are then used as an input to a supervised machine learning algorithm. In a final step, generated knowledge from several user interactions is combined into a taxonomy, which forms the basis for the knowledge engineering process. Following that process, NKE produces structured knowledge as a by-product of users navigating through an information system. Navigation-induced Knowledge Engineering by Example thereby serves several purposes at the same time; it (i) aids users in expressing their informational needs in a structured way (ii) helps them in navigating to resources in a given system and (iii) produces structured knowledge as a result of this process.

Most traditional Knowledge Engineering methodologies heavily rely on a phase-oriented model built on collaboration of a centralized team of domain experts and ontology engineers[21,22,26]. In NKE, web users take the role of domain experts and elicitation is done *en passant* during the navigation process.

The vision of NKE is to enable low-cost knowledge engineering on the largest possible scale - the World Wide Web. The most fundamental consequence of the paradigm is that value is added to data by having a large number of users navigating, using and interacting with it. A reciprocal relation is formed between the informational need of users and the information gained through the created taxonomy. Although structured data is becoming widely available, no other methodology or paradigm – to the best of our knowledge – is currently able to scale up and provide light-weight knowledge engineering for a massive user base. Using NKE, data providers can publish flat data on the World Wide Web without creating a detailed structure *upfront*, but rather observe how structure is created *on the fly* by interested users who navigate the knowledge base.

In summary, this paper makes the following contributions. It

- introduces Navigation-induced Knowledge Engineering by Example (NKE) as a new paradigm for knowledge engineering
- presents a proof-of-concept (HANNE) to demonstrate technical feasibility
- illustrates the new paradigm in an e-commerce context and a query-answering system

The paper is structured as follows: We define Navigation-induced Knowledge Engineering by Example in Section 2 and explain its concepts in detail.

To demonstrate the technical feasibility of NKE, we present HANNE – a Holistic Application for Navigation-induced Knowledge Engineering by Example – in Section 3. HANNE is an Active Machine Learning tool based on Inductive Logic Programming that allows for the extraction of formal definitions (OWL Class Expression) of user-defined concepts based on corresponding examples from arbitrary and possibly large RDF data sets. After we have presented HANNE as a proof-of-concept, we evaluate it in Section4. In Section 5, we review related work on NKE, two fields that we will connect in our work. Finally, we conclude and describe future work.

2 Navigation-induced Knowledge Engineering by Example - A New Paradigm

In this section, we define NKE and give an explanation of the key concepts and requirements related to this paradigm.

Definition: *Navigation-induced Knowledge Engineering by Example is the manifestation of labeled examples by interpreting user navigation combined with the active correction and refinement of these examples by the user to create an ontology of user interests through supervised active machine learning.*

When a web site is displayed in a browser, links are presented to the user for selection. Users typically select a subset of these links to navigate to a particular resource or set of resources. However, as web sites are heterogeneous and thus present a multitude of heterogeneous links, it is difficult - if not impossible - to make proper assumptions about the users' informational needs that are driving their underlying navigation behavior. If we, however, constrain our focus to web sites serving homogeneous content, such as a list of products, people or bookmarks, it becomes easier to analyze the goal of a user more clearly.

The NKE paradigm focuses on those websites, where objects with some form of defined semantics are available, such as Amazon products or Wikipedia articles. As the user is presented with a list of links to such objects, selecting and clicking on a link can be interpreted as positive feedback. All other links are neglected and can be interpreted as negative feedback. This interpretation is, of course, an oversimplification and often wrong: A user might accidentally click on a link or follow a link and then realize, that the target is not what she was looking for. Furthermore not only the selected item of a list might be of interest, but others as well. In addition, it normally remains hidden to a web system, whether the informational need of a user changes during the course of a visit. As soon as e.g. a product is found, the next user action might be triggered by a different need[3]. In many cases however, especially in more interactive systems, it is feasible to approximate the informational needs of a user by observing his interactions with the system.

Navigation-induced Knowledge Engineering by Example: The NKE paradigm consists of three distinct yet interrelated steps: **(i) Navigation:** NKE

[3] Adding the product to a shopping cart could be a good indicator for such a change.

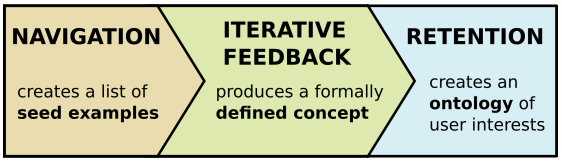

Fig. 1. NKE combines navigational methods with active iterative relevance feedback to create a preliminary ontology.

starts by interpreting navigational behavior of users to *infer* an initial (seed) set of positive and negative examples. **(ii) Iterative Feedback:** NKE supports users in *interactively refining* the seed set of examples such that the final set of objects satisfies the users' intent. and **(iii) Retention:** NKE allows users to *retain previously explored sets of objects* by grouping them and saving them for later retrieval. Thus, the idea of NKE is to use clues from navigational behavior of users in a given system to infer a seed set of positive and negative examples that are later refined interactively by users to advance towards their search goal. This set of examples is later used to infer semantic structures in an active machine learning task.

In the following, we will formulate the underlying requirements related to the three steps in greater detail:

(i) Navigation: *The first requirement for NKE is the ability of a system to approximate the informational need of a user and produce positive and/or negative examples.* Many ways of approximating users' informational needs can be envisioned and are deployed in a multitude of traditional recommender systems. One way of approximating users' needs was followed in the DBpedia Navigator, an early prototype by Lehmann et al. [20]. The DBpedia Navigator could be used to browse over Wikipedia/DBpedia articles. Each viewed article was added automatically to the list of positive examples. A user then could review this list and decide to move entries to the list of negative examples, instead. Another well-known recommender system, which is based on user interaction, is the Amazon.com sales web site. Each view of a product is remembered and statistically analyzed to give a wide variety of personalized suggestions[4]: "More Items to Consider", "Customers with Similar Searches Purchased", "Bestsellers Electronics: Point & Shoot Digital Cameras", "The Best Prices on the Most Laptops", "Customers Who Bought Items in Your Recent History Also Bought" are some examples. The most prominent distinction, however, is the clear lack to explicitly give feedback and refine the presented recommendations.

(ii) Iterative Feedback: *The second requirement for NKE is to support the user in actively managing the list of examples to steer the learning process.* In NKE, the user expresses her informational need by creating a list of positive and negative examples. Although the initial list is gathered automatically by a system as an interpretation of navigational behavior, a chance for correction and iterative refinement is given at a later stage. With this requirement, the

[4] Taken from the frontpage of `http://amazon.com` accessed on Oct, 13th 2010

paradigm gives control to the user, who can actively model her search inquiry based on a seed list of examples. Examples selected by users can be seen as a gold standard of labeled data for active machine learning and the learning result can be used to suggest more objects for labeling.

(iii) Retention: *The third requirement for NKE is to enable the user to sufficiently refine and review the learned result, and let her save it for later retrieval.* Retention is a critical part of the NKE paradigm. After the phase of iterative feedback is concluded, the user has to be able to judge whether the learning result matches her needs and is worth saving. To be able to re-use the saved concept, NKE requires users to assign a name to it. The philosophy is straight-forward: A concept, which is saved by one user to ease further navigation, is likely to be useful to other users as well. As we will see later, the saved concepts will form a taxonomy of user interests, which can be directly exploited as navigational suggestions. Also, the created taxonomy can be considered a raw material, which can be facilitated into a full-fledged domain ontology at low cost.

In the following, we explain the concepts involved in NKE in more detail:

Knowledge Source: NKE requires objects that are represented in a structured form and stored within a knowledge base. The following are typical examples:

- *OWL Individuals* in an OWL knowledge base and their RDF properties.
- saved *bookmarks* on Delicious[5] and their tags.
- *products* on Amazon[6] and the product properties.
- *newspaper articles* in a newspaper database and the article attributes like authors, keywords or links to other articles.

Note that the latter three examples can also be modeled in RDF and OWL[7], which we use for our demonstration. The NKE paradigms can be applied to all formalisms fitting the resource-feature scheme.

Supervised Machine Learning Algorithm: As users choose exemplary resources from the knowledge source, the material for applying a supervised machine learning algorithm is prepared. This algorithm can easily be exchanged and optimized according to the data structure of the knowledge source. In our implementation, we use an algorithm (Inductive Logic Programming) that relies on positive and negative examples, but positive-only or negative-only can be sufficient when using other algorithms. Furthermore, the given examples do not need to be binary in any way and could be assigned a weight, instead. The only limitation is that the algorithm needs to produce learned concepts, adhering to the requirements below.

Learned Concepts: The properties of the learned concepts are central to the NKE paradigm. The learned concepts need to serve as a classifier. This classifier can be either binary (retrieving only those resources from the pool that are covered) or assign a weight (e.g. between 0 and 1) to every resource[8]. The retrieved set of resources is called $r_{classified}$ or *extension* of the concept. As each

[5] `http://www.delicious.com`

[6] `http://amazon.com`

[7] For tags, see [13]. For products, see [10].

[8] If the weight is combined with a threshold, the classifier becomes binary again.

learned concept is defined by its extension, they form a partial order by inclusion: Given learned concepts C and D, D is a subconcept of C, iff $r_{classified\ by\ D} \subseteq r_{classified\ by\ C}$. Therefore resources, which are retrieved by a learned concept will also be retrieved by all higher order concepts. The ordering relation is important. As learned concepts can be saved by a user for retention, the ordering relation clearly creates a distinction between user generated data (such as tags, which have no structure per se) and user generated knowledge.

If the classifier is additionally backed by a formalism for an intensional definition, a binary relation can be defined, which should have the same or similar properties as the inclusion relation on the extensions. Naturally, OWL-DL fulfills all the requirements for such a formalism. The subclassOf relation ($\sqsubseteq$) – as it is transitive and reflexive – creates a preorder over OWL class expressions.

Exploratory search with Iterative Refinement: In our approach, learned concepts can be understood in the following way: As the user explores a knowledge base, she is interested in certain kinds of resources, i.e. she tries to find a set of resources r_{target} that matches her informational need such as *All bookmarks on Java tutorials covering Spring* or *All notebooks with more than 2GB, Ubuntu and costing less than 400 euros.* To express her need, she navigates to resources thereby providing a seed subset of examples $r_0 \subset r_{target}$ in iteration 0. During each iteration i (with i ranging from 0...n), the learning algorithm proposes to the user a new set of resources $r_{classified}$ retrieved via the learned concept. The user then selects more resources from $r_{classified}$ and adds them to r_i creating a new set r_{i+1}. This process can be repeated by the user, until she considers the learned concept a *solution* $lc_{solution}$. The standard measures recall, precision and F-measure apply. The learned concept is correct, if $r_{target} = r_{classified}$.

Two basic assumptions underly our notion of exploratory search: 1. the user either knows all the members of r_{target} or she can quickly evaluate membership with the help of the presented information. 2. Furthermore, the user should be able to make an educated guess about the size of r_{target}. NKE therefore requires an informed user, who can judge whether the search was successful. Although this seems to be a hard requirement for a user, we argue that it can be met quite easily in most cases. Albeit, one limitation of the NKE paradigm is that users who do not know how to evaluate candidate results might be more successful with other methods.

We can also identify several reasons why a NKE-based search might fail: 1. a solution $lc_{solution}$ exists, but the learning algorithm is unable to find it 2. a solution does not exist, because the knowledge source lacks the necessary features and 3. the user selects examples that contradict her informational needs.

Generated Ontology: Saving a learned concept plays a central role in NKE. The design of any NKE system should create strong incentives for users to save solutions once they are found. One such incentive is the ability to retain sets or to view or export a subset of resources $r_{classified}$ for later retrieval. Additionally it is necessary that the user assigns a label upon saving. This way a hierarchy of terms is created which forms an ontology for the domain. Such an ontology –

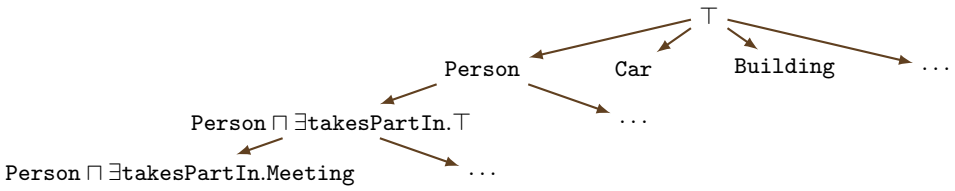

Fig. 2. Illustration of a search tree in OCEL.

as it was created by users to query resources – is a useful candidate to support future navigation or other tasks.

3 Proof-of-Concept: An OWL-Based Implementation

In this section, we introduce and present a proof-of-concept implementation (HANNE[9]), where we employ *DL-Learner* [15] to learn class expressions. We first briefly describe DL-Learner before we explain the HANNE prototype.

3.1 HANNE: Technical Background

DL-Learner[15][10] extends Inductive Logic Programming to Description Logics (DLs), OWL and the Semantic Web. It provides a DL/OWL-based machine learning framework to solve supervised learning tasks and support knowledge engineers in constructing knowledge. In this paper, we use the OCEL algorithm implemented in DL-Learner, because its induced classes are short and readable. OWL class expressions form a subsumption hierarchy that is traversed by DL-Learner starting from the top element ($\top$ in DL syntax or *owl:Thing*) with the help of a refinement operator and an algorithm that searches in the space of generated classes. For instance, Figure 2 shows an excerpt of an OCEL search tree starting from the $\top$ concept, where the refinement operator has been applied for the class expressions $\top$, `Person` etc. The exact details of the construction and traversal of the search tree are beyond the scope of this paper.

When OCEL terminates, it returns the best element in its search tree with respect to a given learning problem. The path leading to such an element is called a refinement chain. The following is an example of such a chain:

$$\top \rightsquigarrow \texttt{Person} \rightsquigarrow \texttt{Person} \sqcap \texttt{takesPartinIn}.\top \rightsquigarrow \texttt{Person} \sqcap \texttt{takesPartIn}.\texttt{Meeting}$$

The way the refinement chains are constructed fits the iterative style of the NKE paradigm. DL-Learner supports the use of SPARQL endpoints, and scales to very large knowledge bases by using an approach that extracts fragments that are small enough for faciliating real time OWL reasoning[9]. The process is sketched in Figure 3.

[9] `http://hanne.aksw.org`
[10] `http://dl-learner.org`

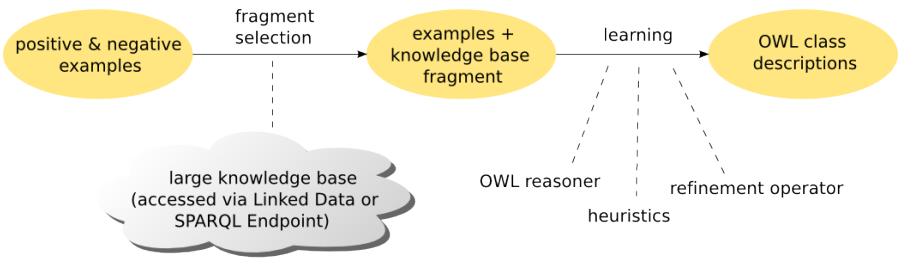

Fig. 3. Process illustration [9]: In a first step, a fragment is selected based on objects from a knowledge source and in a second step the learning process is started on this fragment and the given examples.

3.2 HANNE User Interface

The user interface presented in Figure 4 is a domain-independent implementation of the NKE paradigm, that works on any SPARQL endpoint. For our demonstration, we chose DBpedia as an underlying knowledge base, where the defined goal is to find all 44 U.S. Presidents. Naturally, the list of all U.S. Presidents can be found much faster using e.g. Wikipedia[11]. For the long tail of arbitrary information, such precompiled lists are, however, not easily available. With HANNE, users are able to model such lists according to their information needs as a by-product of navigation.

In our demonstration, the user starts searching for "Bush" to create an initial set of examples. She uses the search components of HANNE marked with 1 in Fig. 4 for that purpose. From the retrieved list of instances, she can select George H.W. Bush and George W. Bush as positive and Kate and Jeb Bush as negative examples. This selection forms the seed set of instances for the second phase - iterative refinement - for which the components marked with 2 (Fig. 4) are used. The user initiates this phase by using the "start learning" button.

The iterative feedback is implemented as follows; using our 4 initial examples, `dbo:Presidents` is learned. By requesting the instance matches of the concept (button "matching"), the user can iteratively select more instances as either positive or negative examples and thereby refine the concept. The count of instance matches and the accuracy of the concept is displayed to help the user estimate whether the concept satisfies her navigation needs. After selecting 3 more positive examples (George Washington, Eisenhower, Roosevelt) and 2 more negatives (Tschudi and Rabbani) the concept has been narrowed down to `(dbo:President and foaf:Person)`, which only covers 264 instances out of 3 million DBpedia resources. During the iterative feedback process, HANNE displays related concepts on the right side (marked with 3). These related concepts were saved by other users and are either sub, parallel or super classes of the learned concept. The retrieval of all 44 presidents can be successful in 3 different ways: 1.) the iterative process is continued until all 44 presidents are added to the positive list (successful retrieval of the extension) 2.) the learned concept

[11] `http://en.wikipedia.org/wiki/List_of_Presidents_of_the_United_States`

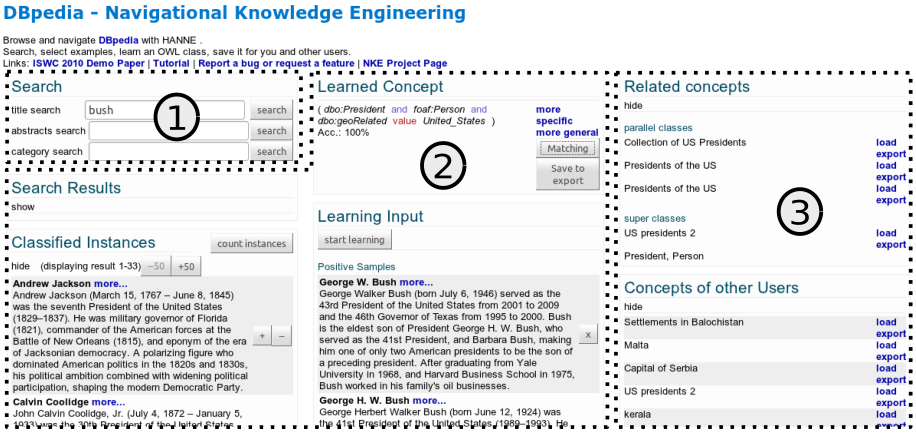

Fig. 4. Screenshot of `http://hanne.aksw.org`: US Presidents in DBpedia.

correctly retrieves all 44 presidents (e.g. `dbo:President and dbo:geoRelated value United_States and dbo:spouse some Thing` retrieves 42 instances) or 3.) a previously saved concept matches the information need (e.g "Collection of U.S. presidents" on the right side).

Solution 1 has created an *extensional* definition and solution 2 an *intensional* definition of the search. Both can be saved and labeled by the user to retain it for later or to export it (either a definition or the SPARQL query to retrieve the instances).

4 Evaluation

Our evaluation scenario is closely tied to the proof of concept implementation presented in the previous section. The evaluation is based on Wikipedia categories, such as *Ships built in Michigan* or *Battles involving Prussia*, which are included in DBpedia. The rationale for evaluating HANNE with the Wikipedia category system are manifold: (1) the categories can be considered a hierarchical structure to more effectively group and browse Wikipedia articles (2) the categories are maintained manually (which is very tedious and time-consuming) and (3) they do not enforce a strict is-a relation to their member articles, which means that the data contains errors from a supervised learning point of view.

From (1) follows that each category corresponds to a potential information need of at least one user, which took the effort to create the collection. For each category (e.g. *Wrestlers at the 1938 British Empire Games*[12]), we can formulate a search task by preceding the category name with "Find all" . Our goal is to evaluate how satisfactory this search task can be addressed with regular search functionality and NKE based methods. (2) is relevant for evaluating the potential

[12] `http://en.wikipedia.org/wiki/Category:Wrestlers_at_the_1938_British_Empire_Games`

of NKE in aiding users in maintaining such a category hierarchy. Finally, (3) allows to assess NKE in realistic noisy scenarios. The evaluation is split in a quantitative and a qualitative part.

The quantitative part will evaluate how well the underlying machine learning approach is able to find all members of a category when compared with a keyword-based search. For the experiment, we used a SPARQL query to retrieve a list of categories from DBpedia, which contained exactly 100 members that had an infobox as well as an abstract property. This selection based on member count is done to avoid bias towards particular domains of categories. The abstract is required for the search engines and the infobox data is required as input for the underlying machine learning algorithm. We envision the following scenario: A user is intensively researching a topic given by the respective category without being able to use the category itself. The main goal of the user is to find **all** members exhaustively and is willing to invest some effort to reach it.

First, we simulate a keyword-based search. The keywords are initially generated from the category names, which we consider as sets of words. In a second step, we remove all stop words and build the power set over the remaining words W resulting in a set of $Q = \{q_1, \ldots, q_n\}$ queries with $n = 2^{|W|} - 1$, ("-1" as we exclude the empty word). Additionally, we optimized the strings by using the singular form and special treatment of hyphens (–) resulting for example in this set of queries: {{*Wrestler, 1938, British, Empire, Game*}, {*Wrestler, 1938, British, Empire*}, {*Wrestler, 1938, British, Game*}, {*Wrestler, 1938, Empire, Game*}, ...}. The search is conjunctive, so usually the queries with many words are very specific but return few results, whereas searching with fewer words finds more but less specific results. For each query set we vary the maximum number of results (*limit*) the user would look at to 20, 100 or 200. Let *results*(q) be the number of results returned by query $q \in Q$ and *category* the articles in the considered category. We calculate the precision and recall in the following manner:

$$union(\{q_1, \ldots, q_n\}) = \left| \left(\bigcup_{i=1}^{n} results(q_i) \right) \cap category \right|$$

$$best(\{q_1, \ldots, q_n\}) = \max_{i=1}^{n} |results(q_i) \cap category|$$

$$inter(\{q_1, \ldots, q_n\}) = \left| \bigcap_{i=1}^{n} results(q_i) \cap category \right|$$

$$R_{max} = \frac{union(Q)}{|category|} \quad R_{best} = \frac{best(Q)}{|category|} \quad R_{avg} = \frac{avg(Q)}{|category|} \quad P_{best} = \frac{best(Q)}{|limit|} \quad P_{min} = \frac{inter(Q)}{|limit|}$$

R_{max} is an optimistic recall estimate taking the combination of all queries into account, R_{best} only considers the query returning most results from the considered category and R_{avg} is a pessimistic estimate, which takes only those results into account, which have been returned by each query. Similarly, P_{best} is the precision of the best query, i,e. the percentage of returned results for the best query. P_{min} considers the precision of the intersection of all queries.

	BC	SO	DL	BC	SO	DL	BC	SO	DL
Limit	20	20	20	100	100	100	200	200	200
R_{max}	0.03	0.07	0.07	0.12	0.22	0.29	0.17	0.28	0.31
R_{best}	0.03	0.06	0.06	0.11	0.20	0.28	0.15	0.25	0.30
R_{avg}	0.01	0.01	0.04	0.03	0.05	0.19	0.04	0.05	0.21
P_{best}	0.13	0.21	0.30	0.11	0.16	0.28	0.08	0.11	0.15
P_{min}	0.04	0.05	0.20	0.03	0.04	0.19	0.02	0.02	0.10

Table 1. Evaluation results for all three approaches (BC = Virtuoso bif:contains, SO = Solr, DL = DL-Learner) with three different limits.

We used two different query engines: (1) The Virtuoso[13] internal word index based on the titles and abstracts, which we ranked based according to the node in-degree following the search approach in the RelFinder tool [6]. (2) a Lucene/-Solr index, which was build from the titles and abstract and also ranked based on node in-degree.

The keyword-based search is now compared to an iterative refinement based on the DL-Learner algorithm used in HANNE. We assume that the user has basic domain knowledge and can name 5 members of the category, which serve as initial positive examples. In a realistic application such an initial set could have been selected from the navigation history, string search or facet-based browsing. Here, the 5 positive articles are chosen randomly from the 100 members. Then 5 negative examples are randomly chosen from the set of members of parallel categories, i.e. (non-equal) categories with the same direct predecessor in the category graph. The OCEL algorithm, implemented in DL-Learner, is then started and the resulting OWL Class Expression is transformed into a SPARQL query with the same *limit* as above. From the result set up to 5 correctly found articles are added to the positive examples as well as 5 negatives. This process is repeated 5 times. We can calculate precision and recall analogously by replacing the set of a set of keyword-based queries with the set of resulting SPARQL queries, one for each of the 5 iterations.

The results in Table 1 show that DL-Learner as a recommender is competitive with the search approaches we implemented and has higher values in all cases. Detailed results are available at `http://aksw.org/Projects/NKE`.

In this experiment, we are not aiming to establish that our implementation is better than a key word based search, as we neglected a plethora of optimization opportunities for string search such as more normalization, ngrams, query expansion techniques, relevance feedback, etc. We think of the searches we implemented as a reasonable baseline, which are actually used in many systems (e.g. RelFinder and DBpedia Lookup). While our searches are based on artificial queries, future research could conduct a similiar evaluation with other queries or actual queries obtained from a live system. In addition, the quality of the results for DL-Learner directly depends on the availability of rich semantics and data quality. In our evaluation, DL-Learner did not find any members for a total of 23 categories. For 11 categories, we could not find any negative examples, as there

[13] Virtuoso is the triple store underlying the DBpedia SPARQL endpoint we used bif:contains for accessing its text index.

were no parallel categories, thus the learned concept was `owl:Thing`. We expect these limitations to become less relevant with the emergence of more knowledge bases that are rich in semantics.

The qualitative part is concerned with the quality of the learned OWL class expression. The authors as experts in ontology engineering (not domain experts) have manually reviewed the results and we will discuss our findings here. We use the excerpt of results shown in Table 2 for discussing four cases of concepts learned.

Single feature concepts In case the categorization depended on a single feature, DL-Learner normally learned the appropriate concept. These learned concepts could be added as an intensional definition to the knowledge base and help to (1) automatically categorize new data as well as (2) find missing properties such as `familia` or `subdivisionName`.

Overly specific concepts In (3) DL-Learner learned an overly specific concept matching only 53 instances. We inspected the data in the endpoint and found that the object values for the property `shipBuilder`, which is highly relevant for this category, were of mixed quality: the property had literal and URIs as objects and the value `Defoe_Shipbuilding_Company` was the one that occurred most frequently (53 occurrences) followed e.g. by `Benton_Harbor,_Michigan` (only 3 occurrences). Note that we used DBpedia 3.6.1 as a basis for our experiments; newer versions might provide higher data quality and completeness. The keyword searches failed completely in this case, as neither "ship", "built" or "Michigan" are selective enough.

Indirect Solution concepts An indirect solution was found for the category in example (4) as no feature (e.g. `champion value US_Open`) was available to construct the concept. The solution reads like a paraphrase and uses the property `subdivisionName`, which is often used by U.S. cities, which are naturally well curated in the English Wikipedia.

Zero member concepts The examples (5) and (6) are two of the categories for which the learned concept is retrieving zero members from the considered category. The category `Northland Region` (5) does not have a clear member of relation, but is rather used as a tag to group all related articles[14], thus members are too heterogeneous. For the category `FC Salyut Belgorod players` (6) a pertinent and specific feature could not be found in the data, thus it is virtually impossible to learn a formalization, which is specific enough.

5 Related Work

5.1 Navigation

Several navigation and knowledge exploration methods can be used in combination with the proposed NKE paradigm.

[14] `http://en.wikipedia.org/wiki/Category:Northland_Region`

Nr	Category	Concept (Manchester OWL Syntax)
1	Pleuronectidae	`familia value Pleuronectidae`
2	Villages in Milicz County	`Village and subdivisionName value Milicz_County`
3	Ships built in Michigan	`Ship and shipBuilder value Defoe_Shipbuilding_Company`
4	United States Open champions (tennis)	`TennisPlayer and (foaf:Person or residence some subdivisionName some PopulatedPlace))`
5	Northland Region	`River and geoRelated value New_Zealand and mouth some Thing`
6	FC Salyut Belgorod players	`currentclub some (chairman some Thing and name some position value Defender_%28football%29 )`

Table 2. Learned concepts for sample categories with limit 100

For knowledge bases with a sufficiently small schema, techniques like facet-based browsing are commonly used. One of those tools is the Neofonie Browser[15]. For very large schemata, facet-based browsing or browsing based on the class hierarchy can become cumbersome and graphical models are used. One example of that uses graphical models is the RelFinder [6]. A user can enter a number of interesting instances and the tool visualises the relationship between those instances as an RDF graph, which can then be explored by the user. Another navigation method is user specific recommendations. Once a user has viewed certain entities, e.g. products, recommender systems can suggest similar products. Often, this is done based on the behavior of other users, but some systems use background knowledge for recommendations. In the simple case, this can be taxonomical knowledge [28], but has recently also been extended to OWL ontologies [20]. The NKE paradigm is based on the latter idea and translates it to the knowledge engineering use case.

Models of user navigation have been successfully used in a range of related domains. For example, in the domain of tagging systems, navigational models [8] as well as behavioral and psychological theories are exploited to evaluate taxonomic structures [7], to assess the motivation for tagging [25], or to improve the quality of emergent semantics [14] and social classification tasks [29]. While navigational models have been applied to improve or evaluate (unstructured) semantics in these domains, they have not been extensively applied to *structured* knowledge bases. This paper sets out to address this gap.

5.2 Knowledge Engineering

Knowledge engineering aims to incorporate knowledge into computer systems to solve complex tasks. It spans across several disciplines including artificial intelligence, databases, software engineering and data mining. Most traditional Knowledge Engineering methodologies heavily rely on a phase-oriented model built on collaboration of a centralized team of domain experts and ontology engineers[21,22,26]. In particular, Pinto et al. [22] characterize the future settings for evolving ontology building as:

Highly distributed: Anyone can contribute more knowledge.

[15] http://dbpedia.neofonie.de

Highly dynamic: Several contributors may be changing knowledge at the same time, with high change rates.

Uncontrolled: There is no control over what information is added, and the quality and reliability of that information. In this case, there will be a lot of noise (positive and negative contributions), and not everybody contributing to the ontology will be focused on the same task or have the same purpose.

In their survey, they argue that future methodologies will need to cope with these properties to be successful and to scale up to the increasing availability of ontologies. In NKE, web users take the role of domain experts and elicitation is done *en passant* during the navigation process. To the best of our knowledge, NKE is currently the only methodology that is not only able to cope with all three properties, but is also designed to exploit them to generate ontologies.

[12] distinguish between the *domain axiomatization* and the *application axiomatization*. Although in NKE, the generated *ontology of user interest* is similar to the mentioned application axiomatization, the DOGMA approach might not be directly applicable to NKE as it uses a domain ontology view to interpret the application model. Ontology matching algorithms could be employed, however, instead of the proposed double articulation to mediate between the application ontologies.

In the following, we briefly discuss work related to Ontology Learning, Knowledge Base Completion and Relational Exploration. Many approaches to Ontology learning rely on Natural Language Processing (NLP) and have the goal of learning ontologies from plain text. Other approaches range from using game playing [24] to Formal Concept Analysis (FCA) and Inductive Logic Programming (ILP) techniques. The line of work which was started in [23] and continued by, for instance [1], investigates the use of formal concept analysis for completing knowledge bases. It is mainly targeted towards less expressive description logics and may not be able to handle noise as well as a machine learning technique. In a similar fashion, [27] proposes to improve knowledge bases through relational exploration and implemented it in the RELExO framework[16].

A different approach to extending ontologies is to learn definitions of classes. For instance, [2] proposes to use the non-standard reasoning tasks of computing most specifics concepts (MSCs) and least common subsumers (LCS) to find such definitions. For light-weight logics, such as $\mathcal{EL}$, the approach appears to be promising. There are also a number of approaches using machine learning techniques to learn definitions and super classes in OWL ontologies. Some of those rely on MSCs as well [4,5,11] while others use so called top down refinement approaches [18,19]. Indeed, the HANNE and Geizhals backends are based on extensions of this work in [19] and [17].

In related research on natural language interfaces, [3] investigate so called intensional answers. For instance, a query "Which states have a capital?" can return the name of all states as an extensional answer or "All states (have a capital)." as an intensional answer. Such answers can sometimes reveal interesting knowledge and they can also be used to detect flaws in a knowledge base.

[16] `http://relexo.ontoware.org/`

6 Conclusions and Future Work

The contribution of this paper lies in the presentation of a new paradigm - Navigation-induced Knowledge Engineering by Example - that conceptually integrates user navigation into a coherent framework for knowledge engineering by the masses. We provided a concise definition of NKE, provide a general proof-of-concept prototype demonstrating its technical feasibility, and show its practical applicability in two different application domains. It is the hope of the authors that the presentation of this paradigm ignites and stimulates further work on the development of navigational approaches to knowledge engineering.

Acknowledgements

This work was supported by a grant from the European Union's 7th Framework Programme provided for the project LOD2 (GA no. 257943).

References

1. F. Baader, B. Ganter, U. Sattler, and B. Sertkaya. Completing description logic knowledge bases using formal concept analysis. In *IJCAI 2007*. AAAI Press, 2007.
2. F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. *J. Applied Logic*, 5(3):392–420, 2007.
3. P. Cimiano, S. Rudolph, and H. Hartfiel. Computing intensional answers to questions - an inductive logic programming approach. *Journal of Data and Knowledge Engineering (DKE)*, 2009.
4. W. W. Cohen and H. Hirsh. Learning the CLASSIC description logic: Theoretical and experimental results. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, pages 121–133. Morgan Kaufmann, may 1994.
5. F. Esposito, N. Fanizzi, L. Iannone, I. Palmisano, and G. Semeraro. Knowledge-intensive induction of terminologies from metadata. In *ISWC*, pages 441–455. Springer, 2004.
6. P. Heim, S. Hellmann, J. Lehmann, S. Lohmann, and T. Stegemann. RelFinder: Revealing relationships in RDF knowledge bases. In *Proceedings of the 3rd International Conference on Semantic and Media Technologies (SAMT)*, volume 5887 of *Lecture Notes in Computer Science*, pages 182–187. Springer, 2009.
7. D. Helic, M. Strohmaier, C. Trattner, M. Muhr, and K. Lerman. Pragmatic evaluation of folksonomies. In *20th International World Wide Web Conference (WWW2011), Hyderabad, India, March 28 - April 1, ACM*, pages 417–426, 2011.
8. D. Helic, C. Trattner, M. Strohmaier, and K. Andrews. On the navigability of social tagging systems. In *The 2nd IEEE International Conference on Social Computing (SocialCom 2010), Minneapolis, Minnesota, USA*, pages 161–168, 2010.
9. S. Hellmann, J. Lehmann, and S. Auer. Learning of OWL class descriptions on very large knowledge bases. *Int. J. Semantic Web Inf. Syst.*, 5(2):25–48, 2009.
10. M. Hepp. Goodrelations: An ontology for describing products and services offers on the web. In A. Gangemi and J. Euzenat, editors, *EKAW*, volume 5268 of *Lecture Notes in Computer Science*, pages 329–346. Springer, 2008.

11. L. Iannone, I. Palmisano, and N. Fanizzi. An algorithm based on counterfactuals for concept learning in the semantic web. *Applied Intelligence*, 26(2):139–159, 2007.
12. M. Jarrar and R. Meersman. Formal ontology engineering in the dogma approach. In R. Meersman and Z. Tari, editors, *ODBASE*, pages 1238–1254. Springer, 2002.
13. H. L. Kim, S. Scerri, J. G. Breslin, S. Decker, and H. G. Kim. The State of the Art in Tag Ontologies: A Semantic Model for Tagging and Folksonomies. In *Proceedings of the 2008 International Conference on Dublin Core and Metadata Applications*, pages 128–137, Berlin, Deutschland, 2008. Dublin Core Metadata Initiative.
14. C. Koerner, D. Benz, M. Strohmaier, A. Hotho, and G. Stumme. Stop thinking, start tagging - tag semantics emerge from collaborative verbosity. In *Proc. of the 19th International World Wide Web Conference (WWW 2010)*, Raleigh, NC, USA, Apr. 2010. ACM.
15. J. Lehmann. DL-Learner: learning concepts in description logics. *JMLR 2009*, 2009.
16. J. Lehmann, C. Bizer, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165, 2009.
17. J. Lehmann and C. Haase. Ideal downward refinement in the el description logic. In *Proceedings of the 19th International Conference on Inductive Logic Programming*, volume 5989 of *Lecture Notes in Computer Science*, pages 73–87. Springer, 2009.
18. J. Lehmann and P. Hitzler. A refinement operator based learning algorithm for the ALC description logic. In *ILP 2007*, 2007.
19. J. Lehmann and P. Hitzler. Concept learning in description logics using refinement operators. *Machine Learning journal*, 78(1-2):203–250, 2010.
20. J. Lehmann and S. Knappe. DBpedia navigator. Semantic Web Challenge, International Semantic Web Conference 2008, 2008.
21. M. M. F. López. Overview of Methodologies for Building Ontologies. In *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem Solving Methods (KRR5) Stockholm, Sweden, August 2, 1999*, 1999.
22. H. S. Pinto and J. P. Martins. Ontologies: How can they be built? *Knowledge and Information Systems*, 6(4):441–464, 2004.
23. S. Rudolph. Exploring relational structures via FLE. In K. E. Wolff, H. D. Pfeiffer, and H. S. Delugach, editors, *Conceptual Structures at Work: 12th International Conference on Conceptual Structures, ICCS 2004, Proceedings*, volume 3127 of *Lecture Notes in Computer Science*, pages 196–212. Springer, 2004.
24. K. Siorpaes and M. Hepp. Ontogame: Weaving the semantic web by online games. In *ESWC*, pages 751–766. Springer, 2008.
25. M. Strohmaier, C. Koerner, and R. Kern. Why do users tag? Detecting users' motivation for tagging in social tagging systems. In *International AAAI Conference on Weblogs and Social Media (ICWSM2010)*, Menlo Park, CA, USA, 2010. AAAI.
26. R. Studer, R. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, 25(1-2):161–198, Mar. 1998.
27. J. Völker and S. Rudolph. Fostering web intelligence by semi-automatic OWL ontology refinement. In *Web Intelligence*, pages 454–460. IEEE, 2008.
28. C.-N. Ziegler, G. Lausen, and J. A. Konstan. On exploiting classification taxonomies in recommender systems. *AI Commun*, 21(2-3):97–125, 2008.
29. A. Zubiaga, C. Koerner, and M. Strohmaier. Tags vs shelves: from social tagging to social classification. In *Proceedings of the 22nd ACM conference on Hypertext and hypermedia*, pages 93–102. ACM, 2011.