Knowl Inf Syst (2014) 41:761–792
DOI 10.1007/s10115-013-0678-y

REGULAR PAPER

Bridging structured and unstructured data via hybrid semantic search and interactive ontology-enhanced query formulation

Markus Gärtner · Andreas Rauber · Helmut Berger

Received: 17 November 2011 / Revised: 19 July 2013 / Accepted: 17 August 2013 /
Published online: 5 September 2013

Abstract In this paper, we identify the problems of current semantic and hybrid search systems, which seek to bridge structure and unstructured data, and propose solutions. We introduce a novel input mechanism for hybrid semantic search that combines the clean and concise input mechanisms of keyword-based search engines with the expressiveness of the input mechanisms provided by semantic search engines. This interactive input mechanism can be used to formulate ontology-aware search queries without prior knowledge of the ontology. Furthermore, we propose a system architecture for automatically fetching relevant unstructured data, complementing structured data stored in a Knowledge Base, to create a combined index. This combined index can be used to conduct hybrid semantic searches which leverage information from structured and unstructured sources. We present the reference implementation Hybrid Semantic Search System (HS^3), which uses the combined index to put hybrid semantic search into practice and implements the interactive ontology-enhanced keyword-based input mechanism. For demonstration purpose, we apply HS^3 to the tourism domain. We present performance test results and the results of a user evaluation. Finally, we provide instructions on how to apply HS^3 to arbitrary domains.

Keywords Semantic Web · User interface design · Semantic annotation · Hybrid Semantic Search System · Keyword-based search · Concept-based search · Ontologies

M. Gärtner (✉) · A. Rauber
Department of Software Technology and Interactive Systems,
Vienna University of Technology, Favoritenstr. 9-11/188, 1040 Vienna, Austria
e-mail: gmax@chello.at; gaertner@ifs.tuwien.ac.at

A. Rauber
e-mail: rauber@ifs.tuwien.ac.at

H. Berger
max.recall Information Systems OG, Pulverturmgasse 17/3, 1090 Vienna, Austria
e-mail: helmut.berger@max-recall.com

1 Introduction

Data on the Internet can be roughly categorized into structured and unstructured data. Structured data is data that is organized in a structure, which is searchable in a straightforward manner, making it easy to pinpoint specific information and access it in a fast and concise way. Examples of structured data are information stored in the RDF format and information stored in a relational database. Unstructured data is the opposite. Due to its unstructured nature, it is a time-consuming task to pinpoint specific information and make effective use of it. Examples for unstructured data are plain text documents, HTML documents, and messages on Twitter[1] and Facebook.[2] HTML Web pages, the most widespread unstructured data sources, still make up the majority of data on the World Wide Web. However, in recent years, structured data has become more widely available as new content was created, having in mind to make the information readily processable and searchable for machines. This idea was one of the main reasons for the emergence of the Web of Data, the Semantic Web. The Semantic Web can be considered as an extension to the World Wide Web that provides structure information in the form of meta-data. The RDF data format has become the de facto standard to make structured data available on the Semantic Web. Hence, there are still two different data sources that coexist : the World Wide Web, which mainly consists of unstructured data, and the Semantic Web, which enriches the World Wide Web with structured data.

Given that the World Wide Web has been grown ever since the early nineties, it holds an enormous amount of knowledge that is scattered as unstructured data over billions of sites. However, even though the Semantic Web cannot compete with the World Wide Web in terms of pure amount of data, it has the significant advantage that its data can be accessed and searched more concise due to its structured nature. Therefore, it would be desirable to bridge the structured data of the Semantic Web with the unstructured data of the World Wide Web to get the best of both worlds. As it is not feasible to manually transform existing unstructured data on the World Wide Web to structured data suitable for the Semantic Web, several automatic approaches have been proposed [25,27,30] that use heuristics and learning techniques to discover relevant entities in text and create meta-data in the form of annotations. These approaches use structured data from relational databases or Triples Stores to assist the extraction and annotation process [22]. The generated meta-data can be used to augment unstructured data on the World Wide Web and help to access it in a concise and efficient way.

The two most common search approaches are the keyword-based approach for unstructured data and the concept-based approach for structured data. For some time, these two approaches have been research separately [15,29,31], but only recently researchers started to explore the possibilities of combining structured and unstructured data and search them conjointly [4,5,14,34].

However, state-of-the-art hybrid search systems and semantic search systems still suffer from a couple of major issues. First, many of these systems lack in usability, preventing the average Internet user from making efficient use of them. This results in a reduced acceptance of these types of systems. The usability and acceptance of a search system is highly related to the used input mechanisms [16]. Several different input paradigms are used by traditional keyword-based search engines and concept-based semantic search engines. The prevalent keyword-based Web search engines use a clean and intuitive interface that consists of an input text field and a search button. Advanced input options are mostly hidden and are only presented to the user on demand. In contrast, there are several semantic search engines,

[1] http://twitter.com/.

[2] http://www.facebook.com/.

which either use a custom input mechanism [3] or can only be accessed via high-level query languages such as SPARQL [26], SeRQL [6] or other RDF-based query languages [7]. The major drawback of semantic search engine user interfaces is that either they are complex and somewhat overcrowded with various input fields or that users need to know a rather complex high-level query language, preventing the average Internet user from using the semantic search engine efficiently. Users may need a considerable amount of time before they get used to the input mechanisms of semantic search engines.

Second, the majority of semantic search systems and hybrid search systems do not use result ranking. Result ranking is common practice in purely document-based search systems, where the most prominent ranking mechanisms use variations of the Vector Space Model and *tf-idf* weighting [28]. In contrast, semantic search systems hardly employ ranking mechanisms to sort their results.

Finally, specifically hybrid search systems face the challenge of acquiring suitable unstructured data that complements the structured data stored in their Knowledge Bases (KBs). In general, this data suits the purpose of extending the knowledge in the KB and complementing it when only sparse structured data is available. However, identifying and acquiring suitable documents containing unstructured data is a tedious manual task, which can be automated to a certain degree.

1.1 Our contribution

To address the lack of usability and the resulting drop in user acceptance, we introduce a novel input mechanism for hybrid semantic search that combines the clean and concise input mechanisms of keyword-based search engines with the expressiveness of the input mechanisms provided by semantic search engines. The average Internet user is not willing to learn a specific query language such as SPARQL or SeRQL to make use of a semantic search system. Even though systems have been introduced that transform natural language into SPARQL queries [21], the average response times of these systems are very high and not comparable to response times of keyword-based Web search engines, which have response times that are fractions of a second. In addition, the proposed interface can be used to interactively formulate queries without prior knowledge of the underlying ontology. To the best of our knowledge, no other semantic search engine offers this possibility. Furthermore, we use the graph-like queries generated by the interface to facilitate result ranking based on the user's query formulation and relevant data in the KB and document index.

We propose a supporting system architecture to address the issue of acquiring suitable documents that contain complementing unstructured data to the structured data stored in the KB. The used mechanisms leverage the information richness of the World Wide Web to automatically identify and acquire additional information. The acquired information is enriched with meta-data from the KB to bridge structured and unstructured data and facilitates hybrid semantic search.

To evaluate our approach, we have created the Hybrid Semantic Search System (HS^3), which seeks to bridge structured and unstructured data to facilitate efficient hybrid semantic search by combining keyword-based and concept-based search approaches. The system uses a fully automated process to (i) fetch relevant unstructured data from the World Wide Web by using an arbitrary ontology and KB to complement structured data stored in the KB, (ii) annotate this information with meta-data from the KB, and (iii) create a combined index that facilitates hybrid semantic search. The system provides an interactive ontology-enhanced keyword-based input mechanism that assists the user by formulating her search queries. This

input mechanism enables the user to create expressive search queries without prior knowledge of concepts, properties, the structure of the ontology or any instances contained in the KB.

We start by presenting related work in Sect. 2. In Sect. 3, we give an overview of HS^3's architecture and its components, followed by a presentation of the combined index structure, the hybrid search approach, and the ranking mechanism. In Sect. 4, we showcase the functionality of the interactive ontology-enhanced keyword-based input mechanism. In Sect. 5, we present the application of our approach to the tourism domain followed by performance test results of HS^3. In addition, we discuss how our approach can be applied to different domains and present the results of a user evaluation of the interactive ontology-enhanced keyword-based input mechanism. We conclude the paper in Sect. 6, followed by an outlook on future work.

2 Related work

Bhagdev et al. [4] have created K-Search, one of the most promising hybrid search implementations, which seeks to combine searches upon structured and unstructured data. In their approach, unstructured data is used as a substitute for structured data whenever no structured data is available. The absence of structured data may be the case due to a lack of suitable concepts in the used ontology or inaccurate Information Extraction (IE) processes. Bhagdev et al. [4] and Bikakis et al. [5] have shown that the combination of keyword-based and concept-based search results in a higher precision and recall, compared to precision and recall of the individual approaches. Furthermore, a user study was conducted that showed that the contestants preferred the combined approach over the pure keyword-based and concept-based approaches due to its expressiveness. However, K-Search uses two separate indexes for the hybrid search and combines the result afterward via result intersection. The keyword-based part of the query is issued against a document index generated by SolR,[3] and the concept-based part of the query is issued to a Sesame Triple Store.[4]

Castells et al. [7] developed a semantic search system that is capable of performing keyword-based and concept-based searches. Their main intent was to develop a semantic search system that retains precision and recall of keyword-based search when information in the KB is incomplete or missing. In addition to the system's KB, a document repository is used in which every document is annotated with at least one entity from the KB. A Vector Space Model is used to rank search results and to overcome the specific characteristic of unstructured information in a structured information system. The system uses the RDF query language (RDQL), which is a graph-based query language for RDF. Apparently, RDQL is the only way to interact with the system. A disadvantage of the system is that it may take up to 30 s until a result is presented to the user. The semantic annotation platform KIM developed by Popov et al. [25] provides also a semantic search functionality. However, KIM's main area of application is the extraction and annotation of entities in documents. Kim uses GATE [8], a Natural Language Processing (NLP) and Information Extraction (IE) platform, for Information Extraction. The platform is equipped with an upper-level ontology, namely the KIM Ontology (KIMO), and a KB that are used for further extraction of knowledge from documents.

Bast et al. [3] developed a semantic search system named ESTER (Efficient Search on Text, Entities, and Relations) that makes use of keyword-based search to speed up the retrieval

[3] http://lucene.apache.org/solr/.

[4] http://www.openrdf.org/.

process. The system can be used with an arbitrary ontology and corresponding KB. Nevertheless, only the usage of the YAGO [32] ontology and a KB created from semi-structured information, extracted from Wikipedia, has been documented so far. The system uses a semi-supervised method that extracts instances and their relation from the semantic information encoded in Wikipedia links and stores them in the KB. ESTER uses so-called artificial words that are created from the entities contained in the KB and annotated to the documents in the document corpus. The successor of ESTER, Broccoli [2], offers a more sophisticated user interface that gives users the possibility to formulate their queries as query trees. Similar to ESTER, Broccoli is capable to search for words, concepts, instances, relations, and a combination of these and still focuses on semi-structured information, extracted from Wikipedia, and the YAGO ontology.

Kasneci et al. [19] developed a semantic search system named NAGA. NAGA, similar to ESTER, uses the YAGO ontology [32], but is not limited to knowledge extracted from Wikipedia. NAGA's KB contains data derived from a number of semi-structured and unstructured Web sources such as Wikipedia and the Internet Movie Database (IMDB).[5] The system comprises a KB, Information Extraction tools, a query and a ranking unit as well as a User Interface for casual and expert users. NAGA's query language syntax and semantics have been derived to a great extend from SPARQL. NAGA uses a scoring model to perform ranking of the result entries. Every result entry is scored by aspects such as confidence, informativeness, and compactness. AVATAR is a semantic search engine that is based on a database approach [18]. The aim of AVATAR is to explicitly model a user's intent encoded in a keyword query by using annotations. Documents are processed via a processing workflow and annotated with references to entities in the KB. Annotators store annotations in a structured data store that has been realized as a thin layer on top of a commercial DBMS.

Moreover, Mukherjea [23] presented a system named SSE that is capable of identifying Web pages on the World Wide Web that are relevant to a certain topic by using a focused crawler. The system implements an algorithm that groups physical domains into logical Web sites by analyzing their domain names and IP addresses. The proposed algorithm helps to eliminate the tightly knit community (TKC) effect which is a problem with Kleinberg's algorithm. Bikakis et al. [5] proposed GoNTogle, a framework for document annotation and retrieval, which is able to overcome restrictions of keyword-based and concept-based search by using a hybrid search strategy. The framework uses the strength of the concept-based approach to compensate the weakness of keyword-based approach and vice versa. However, according to the description of the semantic search approach, only concept annotations are considered during the semantic search process, but relations (properties) between concepts are ignored. Giunchiglia et al. [14] proposed concept search, which shares similarities with the work of [4], but does not use separate indexes for concept-based and syntactic (keyword-based) search. To integrate the two search approaches, concept search substitutes words from the keyword-based approach with suitable concepts. Experiments conducted with the TREC dataset indicated that concept search performs better than the pure keyword-based search.

Fernandez et al. [10] presented a system that consists of a semantic unit, which uses a natural language query to retrieve semantic data from a KB, and a document retriever that provides a ranked list of documents annotated with semantic data to the user. HS^3 provides this functionality as well, but queries are formulated via SERQL. Similar to other systems that use semantic information for the keyword-based search, the hybrid approach performs better compared to the pure keyword-based search approach. Jalali and Matash Borujerdi [17] introduced a mechanism named "concept-based pseudo-relevance feedback" which uses a

[5] http://www.imdb.com/.

hybrid retrieval algorithm to discover the relevance between queries and documents by using combinations of keyword-based and concept-based approaches. The presented approach uses top ranked results of semantic retrieval to expand queries with concepts, which are used to re-calculate similarities and create the final result ranking. Fodeh et al. [11] presented a method to cluster documents by using semantic features. They showed that clustering can be improved by identifying polysemous and synonymous nouns. They identify so-called core semantic features, by using a "unsupervised" information gain measure, which can be used to reduce the dimensionality of the feature set but still maintain good clustering results.

Kaufmann and Bernstein [20] evaluated the usability of natural language interfaces that are used to interact with KBs. Even though the main advantage of natural language interfaces is the intuitive and familiar interaction with semantic search systems, the evaluation has shown that users need to know what is possible to ask beforehand to effectively use the system. The knowledge of possible input phrases and the terminology is crucial to interact with the system. Kaufmann and Bernstein introduced the Habitability Hypothesis, which proposes that query interfaces should not overly control the user with an excessively formalistic language but rather impose some structure on the casual user to guide her during the query formulation process. This hypothesis perfectly fortifies our intention behind the interactive ontology-enhanced keyword-based input mechanism presented in Sect. 4. Zenz et al. [35] presented QUICK, a system for incremental query construction that assists the user in constructing semantic queries from keywords. However, due to the incremental approach of QUICK, users may need to issue several queries to the system until they are able to formulate their actual intend into a final semantic query. Furthermore, users still need a certain understanding of how to construct semantic queries which makes the usage of the interface not intuitive to the average Internet user. Schreiber et al. [29] presented an interesting system named the MultimediaN E-Culture demonstrator, which supports keyword-based queries on an annotated cultural-heritage document and image collection. The MultimediaN E-Culture demonstrator uses a JavaScript, HTML/XML, and CSS-based interactive User Interface, that offers instant concept and instance suggestions to the user's input. The user may choose among different suggested concepts or instances to formulate a query that is issued to the system. However, it is not clear whether the system can cope with complex queries that contain multiple concepts, instances, keywords, and their relations.

In Table 1, a summary of features and characteristics of state-of-the-art semantic and hybrid search systems is given. Our approach tries to maintain their advantages and overcome some of their limitations. With the creation of the Hybrid Semantic Search System HS^3, we seek to overcome the issue that some hybrid search systems such as the one presented by Castells et al. [7] can only be used by knowing complex query languages that are not applicable for the average Internet user. ESTER, which tends to be one of the most mature and fastest hybrid search system, can be improved by incorporating a more suitable input mechanism. Furthermore, hybrid search systems such as [7] suffer from the problem that the execution of queries may take up to 30 s on a corpus of approximately 140K documents until a result is presented to the user. With HS^3, we seek to return results of complex queries within 1 s on a corpus of equal size. Many search systems state that they can be used with arbitrary ontologies. Yet, all of them have just proven their application with a specific ontology. The architecture of HS^3 is designed to work with arbitrary ontologies, and we will apply HS^3 to two distinct domains. Finally, to the best of our knowledge, none of the presented semantic search systems provides an end-to-end process that includes all steps that are necessary to automatically build a document corpus from Web sources, a KB, and a combined index to conduct hybrid searches upon. With HS^3, we seek to overcome this shortcoming by introducing a system that implements a complete end-to-end process.

Table 1 Features and characteristics of state-of-the-art semantic and hybrid search systems

System name	Features	Characteristics
Vector Space Model-based semantic search system	Can cope with missing information in the KB by falling back to a keyword-based search approach	Uses a strict Boolean model to perform searches
	An arbitrary domain ontology can be used	User can only use RDQL to interact with the system
	System can use inference techniques to extract implicit knowledge from the KB	Response times of queries can be up to 30 s
ESTER	Response times are fractions of a second	Only application with semi-structured data from Wikipedia demonstrated
	Interactive User Interface	Not clear how results are ranked
	Can operate with big datasets consisting of millions of triples and documents	
KIM	Builds upon freely available components such as GATE and SESAME	Tightly coupled to the KIM Ontology and KB
	Uses established standards such as RDF, RDFS and OWL Lite	No complex semantic queries supported out of the box
	Comes with a pre-populated KB holding basic entities and facts using the KIM Ontology (KIMO)	No combined search available out of the box
	Provides very good results on NER tasks with the KIM Ontology	Ranking is based on entity popularity timelines analysis
	System is available for research from the developers' Website	
	Provides API for keyword-based and concept-based search	
NAGA	Uses an advanced scoring model for ranking by incorporating information about confidence, informativeness and compactness	Only application with semi-structured data from Wikipedia demonstrated
	Demonstration system available	Demonstration system presented on Website has high response times even for simple queries
AVATAR	Uses the freely available UMIA framework for IE	Only entity class recognition is performed but no annotation to specific entity instances
	System is specialized in the identification of entities in user queries to derive user's intent	Demonstrated application of the system is bound to a corpus of email messages

3 The Hybrid Semantic Search System HS^3

In this Section, we present HS^3, an automated system that can enrich an arbitrary KB with additional knowledge by fetching relevant information from the World Wide Web to automatically build a document corpus and combined index. This combined index can be search via a hybrid semantic search mechanism that combines keyword-based and concept-based search to bridge structured and unstructured information. We will start by giving a

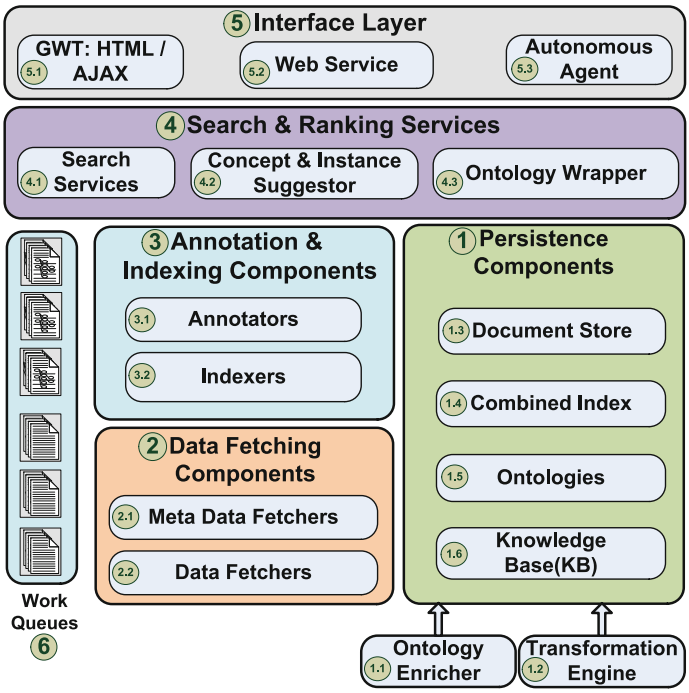

Fig. 1 HS^3 system architecture

short overview of the system followed by an in-detail description of its main components. Figure 1 depicts the component and service-based architecture of the system.

The system is composed of five different types of components, namely the Persistence components (1), the Data Fetching components (2), the Annotation & Indexing components (3), the Search & Ranking services (4), and the Interface Layer (5). The Persistence components (1) are provided with data from the Ontology Enricher (1.1) and the Transformation Engine (1.2). The Ontology Enricher adds textual representations of concepts to the ontology and the Transformation Engine transforms custom data structures to RDF (or OWL) and stores them in the KB. In this context, we refer to the data structures that are not RDF conformed as customer data structures. A textual representation of a concept contains one or more terms that describe a concept. For example, terms such as "canyoning" and "rafting" are textual representations of the concept `Canyoning`. The persistence components comprise the Document Store (1.3), the combined index (1.4), the ontology (1.5), and the KB (1.6). In contrast to other work in which the terms KB and ontology are used interchangeably, we explicitly distinguish between the KB and the ontology. The ontology defines concepts, their properties and relations and the KB includes instances of these concepts and their properties. The ontology corresponds to the TBox and the KB to the ABox as defined in [1]. The ontology therefore only holds a couple of entities (concepts), whereas the KB may hold millions of entities (instances of concepts).

The Document Store holds copies of fetched documents and their respective meta-data. Annotations of documents are stored in the Document Store and can be modified or extended by Annotators (3.1) or Indexers (3.2). The work queues (6) are managed by so-called Work Queue Managers which distribute work to the components of the system. Any component

exposes services that can be used by other components. Metadata Fetchers (2.1) and Data Fetchers (2.2) read data from the KB and write fetched document data to the Document Store. The Metadata Fetcher is used to fetch meta-data for instances of concepts that are stored in the KB. This information is subsequently used by Data Fetchers to fetch data such as HTML documents and store them in the Document Store for further processing.

The Annotation & Indexing components use data that was fetched and filtered by the Data Fetching components. Annotators use the ontologies and the KB to annotate documents in the Document Store. Indexers operate on the ontology, KB, and the Document Store. The system includes a Semantic Indexer, which is part of the Indexer component, out of the box, but custom Indexers can be added to the system as well. The Semantic Indexer creates a combined index that consists of a full-text index and a concept index that holds concepts, instances, their relations, and their textual context within documents.

The system is service-oriented and multi-threaded to support parallelism and facilitate scalability. Services are grouped into system services and consumer services. System services are solely communicating with other system services. Consumer services are accessed from other systems via Web Services or by end-users via a Graphical User Interface. Additional instances of a service can be instantiated if the workload increases. As system services and consumer services are not services themselves and only use for grouping purpose, they are not depicted in Fig. 1. The Search & Ranking services hold consumer services such as the search services (4.1), the Concept & Instance Suggestor (4.2) and the Ontology Wrapper (4.3), which are accessed via the Interface Layer.

The system includes a default search service, namely the Semantic Search Service, which is part of the Search Services component. However, it is possible to replace the default search service with a custom implementation. The Concept & Instance Suggestor service can be used to get all concepts and instances of the KB that match a certain textual representation. The Ontology Wrapper is a service to access all ontologies that are managed by the system. The Interface Layer can make use of any consumer service. It offers three different types of interfaces: a GWT-based Web User Interface (5.1) which offers the interactive ontology-enhanced keyword-based input mechanism to assist users in formulating search queries, a Web Service Interface (5.2) that provides access to the Semantic Search Service and a communication facility for autonomous software agents (5.3), which is realized as Web Service as well. In the following, we concentrate on the Indexing and the Search & Ranking components in more detail.

3.1 Indexing component

The combined index structure, which is based on SIREn [9] and generated by the Indexing Component, can be used to conduct efficient hybrid searches without the need for two separate indexes. The Indexing Component operates on the annotated document corpora in the Document Store. For scalability reasons, multiple Indexers can access the Document and Triple Store simultaneously and modify the combined index in parallel. To generate the combined index, Indexers analyze the annotations and the text of a document. The mechanism is described as pseudo code in Algorithm 1 and explained in the following paragraphs.

An Indexer loads an annotated document from a document corpus and creates a new Lucene[6] document. For every annotation in the document it creates an *Index Graph* that consists of *index graph tuple* objects. The *index graph tuple* objects resemble a set of rows with the columns `subject.concept`, `subject.instance`, `predicate.property`,

[6] http://lucene.apache.org/.

```
Input: Annotated document G
iW = getIndexWriter();
L = new LuceneDocument();
foreach Annotation i in G do
    cI = i.getConceptInstance();
    if cI already indexed for L then
        continue with next i;
    end
    iGTP = getIndexGraphTuples(cI);
    if iGTP == null then
        iGTP = loadIndexGraphTuples(cI);
    end
    iGRC = getIndexGraphRootConcept(i);
    r = getInstanceRelevance(cI,G); L.addField(iGRC.name + "_" + r,iGTP);
    iC = getInstanceContext(cI);
    L.addField(iGRC.context + "_" + r,iC);
end
dS = generateSummary(G);
L.addField(iGRC.summary,dS);
iW.index(L);
```

Algorithm 1: Simplified version of the indexing mechanism

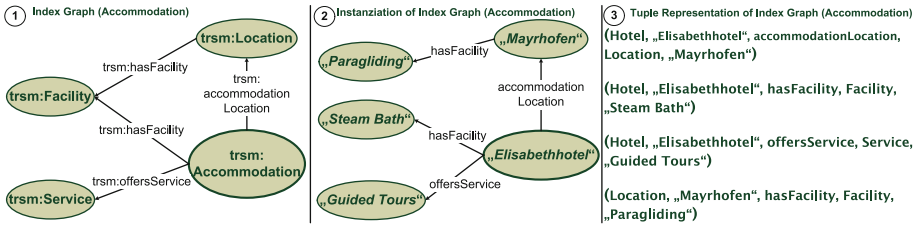

Fig. 2 Index graph definition, index graph instance, and tuple representation

`object.concept` and `object.instance`, depicted in the $IndexGraph\langle IndexGraphRootConcept\rangle \ldots \langle Rel.Marker\_1\rangle$ Section in Fig. 3. Every row holds information that is usually encoded in a RDF statement consisting of subject, predicate and object. But, in addition, the concept type of the subject and the object are also encoded in the *index graph tuple*. For example, the information that the hotel Elisabethhotel provides a steam bath is encoded as *index graph tuple* as follows: {{`Hotel`}, {"Elisabethhotel"}, {`hasFacility`}, {`Facility`}, {"steam bath"}}.

In RDF, this would be accomplished by using two additional RDF statements to declare the type of the subject and object via the `rdf:type` property. For efficiency reasons and to maintain a smaller index, this information is encoded in a single tuple in the combined index. The first column of an *index graph tuple* object holds the concept type of the subject (in the example `Hotel`), the second column holds the actual instance of the subject (in the example "Elisabethhotel"), the third column holds the property (in the example `hasFacility`), the fourth column holds the concept type of the object (in the example `Facility`), and the last column holds the instance of the actual object (in the example "steam bath").

For demonstration purpose, consider the *Index Graph* definition marked with the number 1 in Fig. 2. An actual instance of this *Index Graph* is marked with the number 2 in Fig. 2. Finally, marked with the number 3 in Fig. 2, the corresponding tuples of the *Index Graph*'s instance are listed.

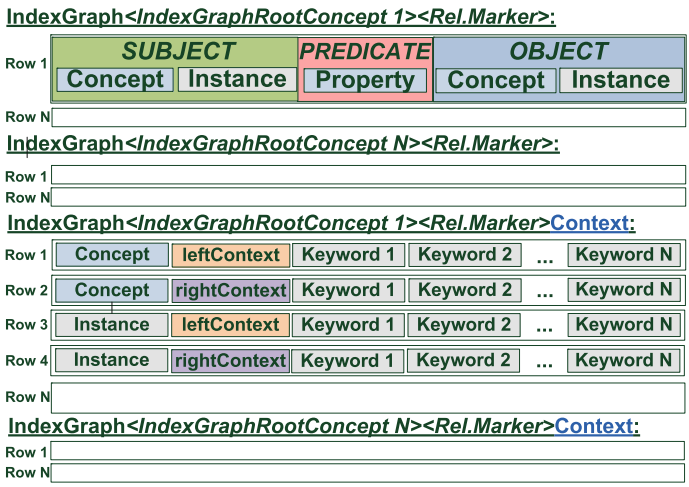

Fig. 3 Combined index structure

Any RDF graph is converted into this structure before it is indexed via SIREn. A document may refer to multiple *Index Graphs*, in case the root nodes of these *Index Graphs* are referenced by annotations in the document. Furthermore, an *Index Graph* may be referred to by multiple documents if its root node is referenced by annotations in multiple documents.

Every concept in the ontology may carry an *Index Graph* that defines a sub-graph of the ontology that should be indexed for this specific concept. This graph is used by the search mechanism to conduct searches for this concept or instances of this concept. The *Index Graph* is attached as a RDF Construct Query to the concept. When loading *index graph tuples* the Indexer issues the RDF Construct Query to the Triple Store and transforms the result into the *index graph tuple* structure. In addition, the Indexer extracts the context of the current instance. The context of an instance is defined as the terms which surround the instance's annotation in a specific document. All terms to the left and to the right of an instance's annotation, that have a distance smaller than a specified threshold, are returned as context. These terms are indexed as N tuples, depicted in the Section $IndexGraph\langle IndexGraphRootConcept\_1\rangle \ldots \langle Rel.Marker\_1\rangle Context$ in Fig. 3. For example, if the words "quiet" and "family-friendly" are put in front of an annotation of a `Hotel` the context entry looks as follows : {{`Hotel`}, {`leftContext`}, {"quiet"}, {"family-friendly"}}.

The index is based on a specific naming convention. All *Index Graphs* start with the *Index Graph* URI followed by an *index graph root concept* and a relevance marker. The relevance marker is determined by considering the Annotation Score of the specific instance.

For example, the *index graph root concept* for the concepts `Hotel` and `GuestHouse` is the concept `Accommodation`, which is their super concept in the ontology as well. The reason for this approach is that the *Index Graph* needs to be defined only for a super concept and is inherited to all sub-concepts. Consider a user who is searching for either a hotel or a guesthouse. The retrieval process would look for a match in the *IndexGraph:Accommodation_1* section of the index rather than a section *IndexGraph:Hotel_1* or *IndexGraph:GuestHouse_1*. This mechanism is needed to facilitate an efficient search for approximate matches to the user's query by using ontology knowledge, which is described in more detail in Sect. 3.2. The relevance marker is added to the very end of the section name. It reflects the relevance

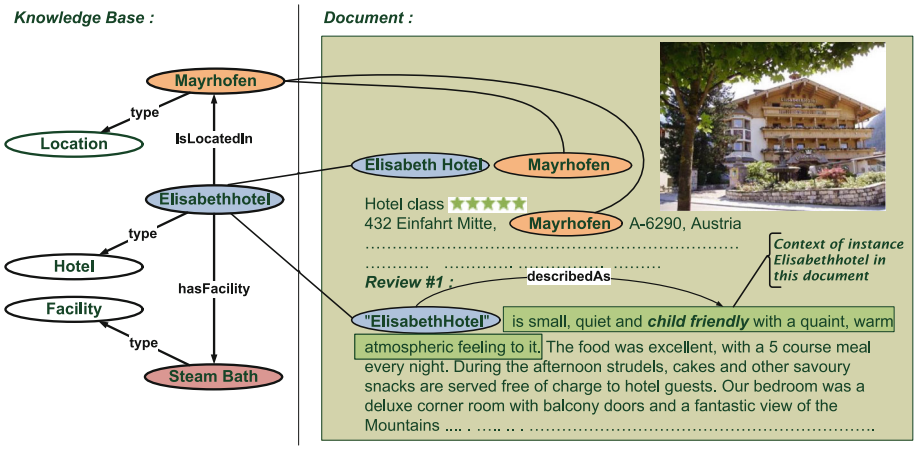

Fig. 4 Supplementation of knowledge in KB and World Wide Web

of the instance, described by the *Index Graph*, in the specific document. The relevance of an instance in a specific document is determined by its Annotation Score, its annotation's position in the document (e.g. title or body) and its annotation frequency in the document.

The relevance marker is used during the search process to restrict searches only to the most relevant instances of a document. This approach helps to reduce noise in the result, because common instances may be mentioned in many documents, even though the documents are mainly about completely different instances. However, instances of lower relevance for a specific document need to be kept in the index, because these are used for approximate matches in case no exact matches can be found. Furthermore, instances of lower relevance might just be of low relevance because the IE processes were not able to extract accurate information. Still, these can be useful in combination with keyword-based searches, which can leverage information that was missed by the IE processes. Finally, the Indexer generates a summary of the document, including the annotations and the context surrounding them.

To demonstrate the advantage of the combined index structure, consider the tourism related search query "child-friendly hotel with steam bath located in mayrhofen" and the instance of an *Index Graph* and annotated document depicted in Fig. 4. This document would be returned and ranked among the top results for the previous query, even though the document does not mention anything about a steam bath nor does the KB hold any information that Elisabethhotel is a child-friendly hotel. However, the KB holds the information that Elisabethhotel is equipped with a steam bath and that it is located in Mayrhofen and the document holds the information that Elisabethhotel is child-friendly. Hence, the document is returned because the combined index amalgamates this information via index graphs and concept contexts. Therefore, the user gets what he was looking for. In case separate indexes would have been used, the document would not have been ranked among the top documents.

3.2 Search & Ranking components

The Search & Ranking components use the combined index generated by the Indexers of the Indexing Component. Instead of just translating the concept-based and combined queries into high-level query languages such as SPARQL or SeRQL, the query is transformed into

tuples and issued against the combined index. Furthermore, keywords which either modify concepts in the query or represent themselves concepts, which are not part of the ontology, are incorporated as tuples as well. A keyword that modifies a concept is a keyword that describes a concept in more detail. For example in the query "child-friendly hotel" the keyword "child-friendly" describes the concept `Hotel` in more detail. By transforming the query into tuples that match the structure of the combined index, a fast retrieval of documents matching the query can be accomplished. During the transformation process the ontology is used to expand the query with sub-concepts and similar concepts, to return approximate results in case no exact matches have been found.

The query expansion mechanism leverages so-called *Realms* to pick only meaningful concepts for the query expansion. A *Realm* holds concepts that are semantically related (e.g. similar concepts). To illustrate the query transformation process consider the query depicted in Fig. 5 "family-friendly guesthouse providing steam bath located in mayrhofen offering rafting". By using the interactive ontology-enhanced keyword-based input mechanism described in Sect. 4, the keyword query is transformed into the graph query depicted in the upper half of Fig. 5. This graph representation is transformed into a tuple query that consists of a disjunction of conjunctions representing the tuples and is issued against the combined index. Algorithm 2 depicts the transformation mechanism, which expects an ordered set of `Resources` as input. The ordered set of `Resources` is generated by the query formulation mechanism. Every `Resource` is either an instance of a concept, a concept, a property or a keyword, which can be related to other `Resources` in the query.

```
Input: Ordered Set OS of Resources
Output: Disjunction List DL of conjunctions
DL = new DisjunctionList();
foreach Resource r in OS do
    type = r.Type();
    if type == Concept then
        rR = r.getRelatedResources();
        tCList = createTupleConjunctions(r,rR);
        foreach TupleConjunction tC in tCList do
            DL.add(tC);
        end

    else if type == Keyword then
        DL.add(r);
    else
        skip current r;
    end
end
```

Algorithm 2: Simplified version of the query transformation mechanism

The transformation mechanism iterates through all `Resources` and checks whether the `Resource` is of type `Concept` or `Keyword`. In case the `Resource` is a `Concept`, related `Resources` in the user's query are identified and the query is expanded with similar `Concepts`. First, to identify all related `Resources`, all related `Resources` of a `Concept` in the query are retrieved, which can either be `Keywords` that modify the current `Concept` or other `Concepts`. Related `Concepts` are always related via a property. Therefore, the generated conjunction is a tuple that consists of the current `Resource's Concept`, the related `Resource's Concept` and the property that connects the two `Resource's Concepts`. In case the related `Resource` is a keyword the tuple will contain the current `Resource's Concept`, a property that states that the `Keyword` belongs to the

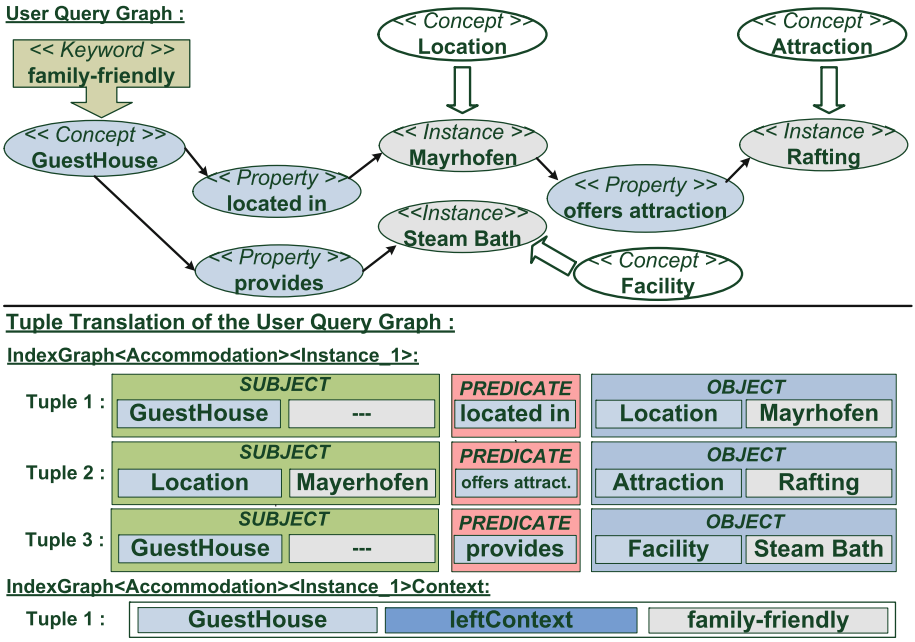

Fig. 5 Graph query translation

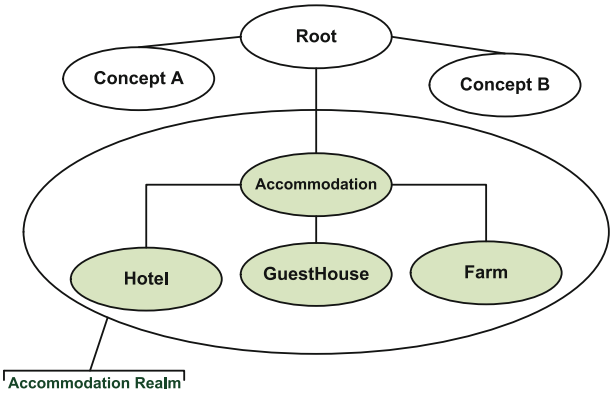

Fig. 6 Accommodation Realm

`Concept's` context and the `Keyword` itself. For example, in the query "child-friendly hotel located in mayrhofen", the string "hotel" is identified as the concept `Hotel`, "mayrhofen" as an instance of the concept `Location`, "located in" as property and "child-friendly" as `Keyword` modifying the concept `Hotel`. Second, having identified the `Resources` and their relations the query is expanded by retrieving similar concepts of those `Resources` that are of type `Concept` by considering their *Realms*. As depicted in Fig. 6 the concept `Hotel` is in the *Accommodation Realm* with the concepts `Accommodation`, `Guesthouse` and `Farm`, where `Accommodation` is the super concept and `Hotel`, `Guesthouse` and `Farm` are similar concepts. However, the concepts `Root`, `Concept A` and `Concept B` are not within the *Accommodation Realm* and are not of relevance for this specific *Realm*. Therefore,

the system uses the *Accommodation Realm* to expand the query and ensures that child-friendly guesthouses and farms that are located in Mayrhofen are also considered during the search process. Pruning is not needed as the system will only consider those `Concepts` that are part of the specific *Realm*.

The resources and their relations are stored in a *tuple conjunction* object, as described in Algorithm 2. The ranking mechanism ranks the documents that contain information matching the user's query. The *tuple conjunction* objects are used in the ranking model to identify and rank objects that are of relevance according to the user's query. Every *tuple conjunction* object is equipped with a score that is used in combination with a document's score to determine the final rank of that document in the result.

3.2.1 Ranking model

The ranking model makes use of four aspects that are encoded in the user's query and the combined index.

1. The type of the resources (instance, concept, or keyword) and their relations in the user's query.
2. The distance between the concepts and instances in the user's query.
3. The occurrence of concepts and instances of the query in the encoded index graphs of the combined index.
4. Context information encoded in the combined index.

First, the type of resources and their relations within the user's query are identified. For this purpose, a pair of adjacent `Resource's` connected via a property are stored in a *tuple conjunction* object. This *tuple conjunction* object is equipped with a base score that is determined by the types of the two `Resources` that are connect via the property. The highest base score 8 is assigned to a *tuple conjunction* that contains two instances of a concept and a property. The next higher base score 4 to a *tuple conjunction* that contains an instance of a concept and a concept. The least base score 1 is given to a *tuple conjunction* that contains just two concepts and a property. Therefore, *tuple conjunctions* which represent the user's information need in the most concise way are assigned the highest score in the query. For example, a *tuple conjunction* that consists of the instance "Hotel Elisabeth", the property `located in` and the instance "Mayrhofen", would express the information need of a user for a specific hotel in a specific city.

However, a *tuple conjunction* that consists of the concept `Hotel`, the property `located in` and the instance "Mayrhofen", expresses the information need of a user in a much broader sense. In that case, the user is not looking for a specific hotel, but rather for an arbitrary hotel that is located in Mayrhofen. Hence, the user is invariant about the ranking of documents describing hotels as long as the described hotels are located in Mayrhofen. Any other hotel that is not located in Mayrhofen is of little interest to the user and is ranked lower. An example for a least specific *tuple conjunction* would be one that contains the concept `Hotel`, the property `located in` and the concept `Near River`. In this case, any hotel that is near a river would be part of the result but ranked arbitrary.

In terms of query expansion, every *tuple conjunction* is extend with all possible concept and instance combinations. For example, the *tuple conjunction* "Hotel Elisabeth"—`located in`—"Mayrhofen" is extended with the *tuple conjunctions* `Hotel—located in`—"Mayrhofen", "Hotel Elisabeth"—`located in—Location` and `Hotel—located in—Location`. To ensure that documents that contain information, the user was initially

looking for are ranked higher the base scores of *tuple conjunction* that where generated due to the query expansion mechanism are multiplied by 0.5 and therefore contribute less to the overall score of a document. Therefore, in the result, those documents that are about hotels named "Elisabeth" which are located in Mayrhofen are ranked higher than those documents that are about other hotels in Mayrhofen, hotels that are named "Elisabeth" but are not located in Mayrhofen and any hotels that are located in any location.

Second, the position of a *tuple conjunction* in the query graph is used to adapt its base score. The farther away a *tuple conjunction* is from the root *tuple conjunction* of its query graph, the lower is its impact. This is realized by multiplying the base score of this *tuple conjunction* N times 0.5, where N is the distance between the root *tuple conjunction* and the current *tuple conjunction*. For example, if a *tuple conjunction* is three *tuple conjunctions* away from to the root *tuple conjunction*, its base score is multiplied by 0.125.

Therefore, those *tuple conjunctions* which are stated first in the query graph are considered as more important than those at the outer border of the query graph. Consider the query graph depicted in the upper region of Fig. 5. In this query graph, the root concept of the graph is `GuestHouse` which is modified by the property `located in` and the instance "Mayrhofen". The instance "Mayrhofen" is modified by the property `offers attraction` and the instance "Rafting". Since "Mayrhofen" modifies the root concept of the graph and "Rafting" only modifies "Mayrhofen", the base score of the *tuple conjunction* that contains the instance "Rafting" will be reduced.

Third, only those documents are considered that contain at least one concept or instance that is mentioned in the user's query. Documents which contain concepts and instances in the document title and body are considered as more important than documents which contain them only in the body. The calculated score of a document is currently multiplied by 10 for every instance and concept that is annotated in the title and the body. In addition, the total count of concepts and instances that are mentioned in the title and in the body are considered as well, boosting those documents that contain many annotations that reference information the user is looking for. The same mechanism is used for keywords that are part of the users's query. Hence, as the combined index stores the semantic *Index Graph* and the full text of a document in the same index it is straight forward to apply a similar logic.

Finally, the ranking model makes use of the context information that is encoded on a per document basis in the combined index. For example if a query contains a *tuple conjunction* that consists of the concept `Hotel`, the property `describedAs` and the keyword "child-friendly", the ranking algorithm considers this as a contextual information. Currently a *tuple conjunction* consisting of a concept, the property `describedAs` and a keyword gets a base score of 8 which is also reduced with respect to the location of the *tuple conjunction* in the user query graph. This means that the algorithm will boost the rank of documents that hold instances of type `Hotel` that are described via the keyword "child-friendly" in their proximity. Currently, proximity is defined as 10 terms to the left and to the right of the corresponding annotation in the document.

For a better understanding of the ranking model, we use a simplified version of a combined index and an example of a user query depicted in Fig. 7 to calculate the score of every document in the combined index given the user query. For simplicity reasons, every document in the combined index contains only one annotation and a related *Index Graph* that is already represented as tuples. Document 1 contains one annotation of `GuestHouse` Strolz and an *Index Graph* with two tuples. Document 2 contains one annotation of `Farm` Walchenhof and an *Index Graph* with one tuple. Document 3 contains one annotation of `GuestHouse` Neuhaus and an *Index Graph* with three tuples and context information with one tuple.

Consider the user query "family-friendly guesthouse with steam bath in mayr-hofen with the possibility to go rafting" which is translated by the system in into the four tuples that are depicted in Fig. 7. First, every tuple in the query is converted into a *tuple conjunction* and the corresponding base score is assigned. *Tuple conjunction 1* gets assigned a base score of 4, *tuple conjunction 2* also a base score of 4, *tuple conjunction 3* a base score of 8 as it contains two instance specifications and *tuple conjunction 4* a base score of 4. As *tuple conjunction 3* has a distance of 1 to the root node, its score is multiplied by 0.5 resulting in a score of 4. Therefore, in this example, every *tuple conjunction* has a score of 4. In the next step, query expansion is performed. For simplicity reasons, we assume that the concept `Accommodation` has only two sub-concepts which are `GuestHouse` and `Farm`. Therefore, three additional *tuple conjunctions* are added where `GuestHouse` is replaced by `Farm`. However, as the concept `Farm` is not what the user was initially searching for, every *tuple conjunctions* that was created as result of the query expansion mechanism is multiplied by 0.5 resulting in a score of 2 for these *tuple conjunctions*.

The expanded and weighted query is used to calculate the scores for every document that contains at least one *Index Graph* that contains one or more tuples which match one or more tuples of the user query. For simplicity reasons, we do not take into account whether the annotation is part of the title or part of the title and the body, nor how often an annotation occurs in the document and also not how many other annotations exist in the document in this example. As document 1 contains an *Index Graph* that contains two tuples that match the query, where each tuple in the query has been assigned a score of 4, the final score of document 1 is 8. As document 2 contains an *Index Graph* that contains one tuple that matches a tuple in the expanded query which has been assigned the score 2, the final score of document 2 is 2. As document 3 contains an *Index Graph* that contains three tuples that match the query, where each tuple in the query has been assigned a score of 4 and one tuple in the context that also matches a tuple in the query which has been assigned a score of 8, the final score of document 3 is 20. Therefore, document 3 is ranked the highest, followed by document 1 which is followed by document 2.

4 Interactive ontology-enhanced query formulation

The User Interface is crucial to the success and acceptance of a system or Website in general. The User Interface of most traditional search engines consists of a single text field which accepts arbitrary text and an optional search button such the search interfaces of Google, Yahoo, or Bing. This simple and clear input paradigm is widely accepted by users of the World Wide Web. Semantic search systems offer a variety of User Interface types such as keyword-based interfaces [3], natural language interfaces [33], graphical query interfaces [36], interfaces that use various HTML form elements for query formulation [4,19] or a combination of these. However, semantic search systems oppose an additional level of complexity, because the user needs to know the terms (concepts) and their relations to formulate expressive queries to make use of a system's knowledge. Depending on the type of interface users need to learn its specifics and have a knowledge of the ontology before they are able to formulate queries. Even though some interfaces might be as simple as a text box, a user needs to know an RDF query language such as SPARQL, SeRQL or even a complicated custom query language to interact with the system.

To the best of our knowledge, ESTER [3], the semantic search system proposed by Castells et al. [7], the system presented by Bhagdev et al. [4] and the system presented by Giunchiglia et al. [14] are the only systems that provide a search service and interfaces that can use a

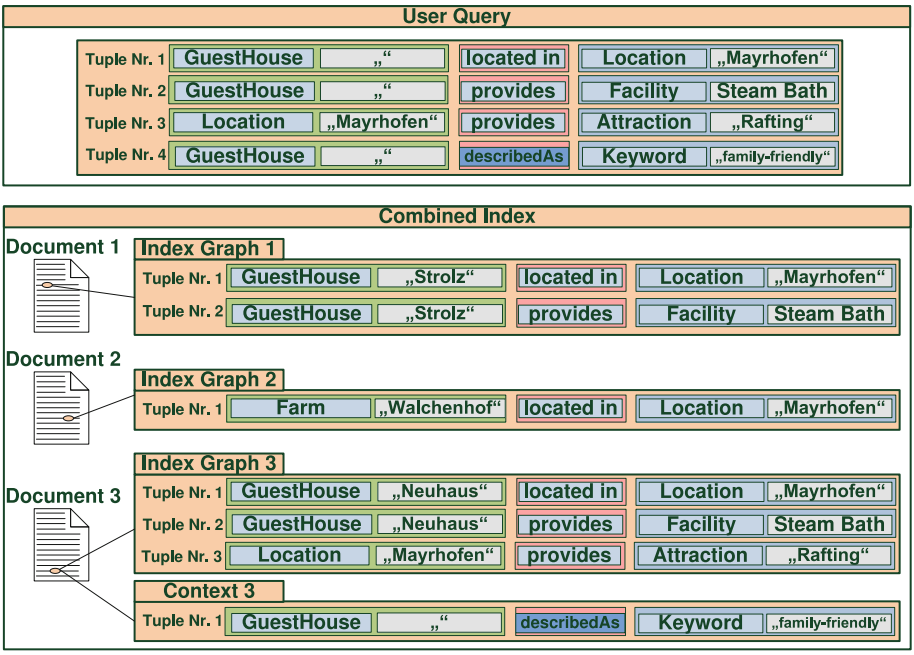

Fig. 7 User query in tuble format and a simple combined index

combination of keyword-based and concept-based search out of the box. However, all of the interfaces suffer from one or more of the following problems: (i) the interface is too complicated for the average Internet user or the user needs considerable time to familiarize herself with the interface, (ii) the user needs beforehand knowledge of the ontology to be able to formulate queries to efficiently access the knowledge of the system, (iii) keywords and concepts are treated as separate inputs.

To overcome these obstacles, we introduce an interactive ontology-enhanced keyword-based input mechanism that uses the established text field input paradigm, combines it with interactive concept, property, and instance suggestion, and helps the user to formulate an ontology-aware query without prior knowledge of the underlying ontology. Therefore, while formulating the query the user familiarizes herself with the structure of the ontology, its concepts, the properties of concepts, and instances stored in the KB. Furthermore, we amalgamate keyword and concept-based input by letting the user enrich the query with keywords. One of the main objectives was to create an interface that lets a user input her query, while it concurrently suggests suitable concept and instance without interrupting the user's input flow. Furthermore, the interface automatically structures the input in a graph-like manner.

In the following, we describe the mechanisms used to realize the interactive ontology-enhanced keyword-based input mechanism and showcase the User Interface by means of a search query from the tourism domain. Part of the User Interface, namely the input text field, is depicted in Fig. 8. The interface relies on three main components. The first component is the OntologyWrapper which is located on the server side. Upon calling the Website of HS^3 the first time, the OntologyWrapper sends a light-weight model of the ontology to the client. This model holds all concepts, all properties, their relations and structural information. The client side model is used to suggest super-concepts or similar concepts when a concept is recognized in the input text field. Furthermore, the model is used to show a pop-up list

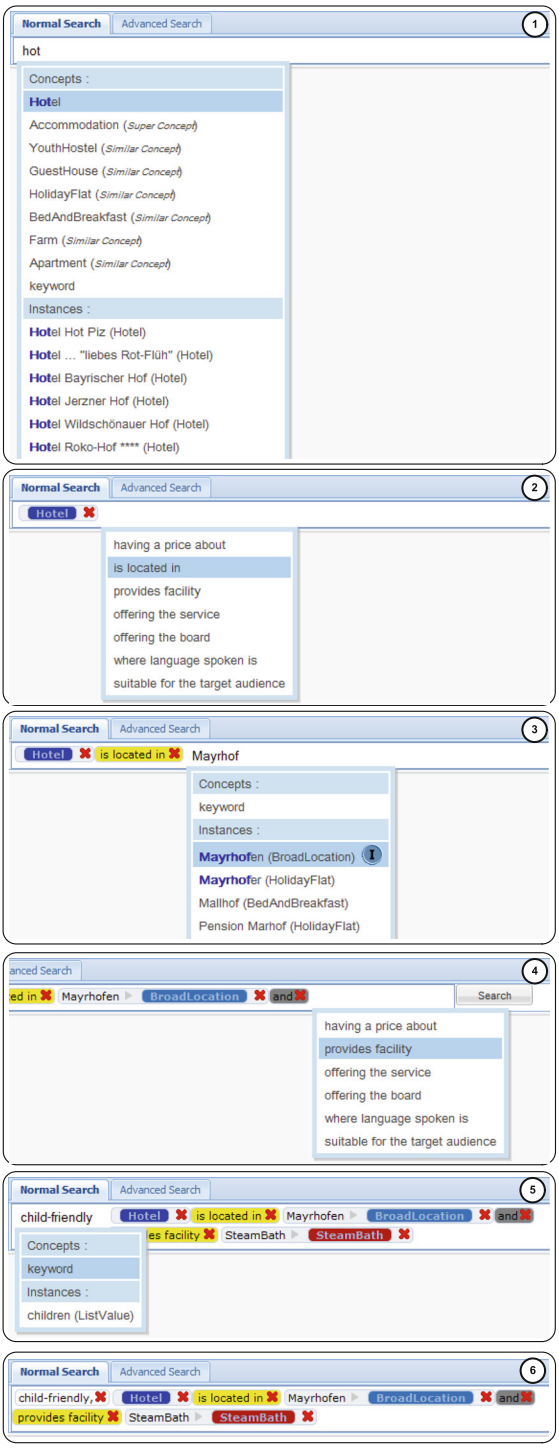

Fig. 8 Interactive ontology-enhanced keyword-based input mechanism

with properties as soon as the user chooses one of the suggested concepts or instances. The second main component is the Concept & Instance Suggestor which resides on the server. The Concept & Instance Suggestor holds an optimized index of all textual descriptions of concepts and instances. In addition, it holds the instance and/or concept URI. Asynchronous calls are issued to the Concept & Instance Suggestor whenever the user hits a key, by using the asynchronous callback functionality of GWT, [7] which return those ten concepts and instances that have a textual description with the smallest Levenshtein [24] distance to the current input term. By using the Levenshtein distance as similarity measurement it is possible to suggest suitable instances an concepts despite any typos in the query.

The final component is the GWT-based User Interface on the client side which displays the user query graphically. Concepts, properties, and instances are displayed within colored oval entities, which can be edited and removed by the user. For a better understanding of the interface, we will showcase it by means of an example query depicted in Fig. 8. In our scenario, the user is interested in accommodations, preferably hotels, which offer a steam bath and are located in Mayrhofen. The user gets presented with the text field and starts to type "hot". As soon as the users starts typing, asynchronous requests including the text field's current input are send to the Concept & Instance Suggestor which returns an ordered set of concepts and instances that have a sub-string which matches the current term or a Levenshtein distance below a defined threshold. In case of the string "hot," the concept and instance suggestions depicted in the first box of Fig. 8 are returned. To display super-, sub-, and similar concepts in the suggestion list, we make use of the client side ontology model and *Realms*. On the client side, we use *Realms* to group end-user relevant hierarchical concepts.

As soon as the user has chosen an appropriate concept or instance, in the presented showcase the concept `Hotel`, the interface shows a pop-up list with the properties of the concept as depicted in box 2 of Fig. 8. It is possible to equip all properties of the ontology with a display string, which is used to display the property in the User Interface. If no display string is defined, the property will not show up in the suggestion list. This mechanism ensures that properties which are not useful for query formulation (e.g. the property that holds the textual representation of the concept) or are only used for internal purposes are not propagated to the User Interface. Having chosen the `located in` property, with the display string "is located in" the user starts typing "Mayrhof" as depicted in box 3 of Fig. 8. This time the asynchronous request to the Concept & Instance Suggestor holds the property information of the query as well. This information is used to rank the instance suggestions according to the concept class that is expected by the property type. In case of `located in` a `Location` concept is expected. Since `BroadLocation` is a sub-concept of `Location` the instance "Mayrhofen" is ranked before the instance "Mayrhofer" which is of type `HoldidayFlat`.

Having chosen the instance "Mayrhofen," the user wants to add that the hotel should be equipped with a steam bath. Therefore, she types "and" and declares the term as operator from the displayed pop-up list. This signals the User Interface that the user is referring to the concept `Hotel` and it uses the client side ontology model to show available properties of the `Hotel` concept. However, the property `located in` is not shown anymore, since the ontology defines that the property `located in` can only be assigned to one `Location`.

Subsequently, the user chooses `provides facility` and picks the "SteamBath" instance as depicted in box 4 of Fig. 8. Since the user is looking for child-friendly accommodations, she adds the keyword "child-friendly" to the query (box 5 of Fig. 8), because the used ontology does not offer a corresponding concept. The final query is depicted in box 6 of Fig. 8.

[7] http://code.google.com/webtoolkit/.

5 HS^3's application to the tourism domain

In the following section, we showcase the application of HS^3 to the tourism domain. We used a tourism ontology based on the Harmonise ontology [12] for the KB. The initial dataset for the KB was generated from trusted information that is stored in a RDBMS. The database, provided by the e-Tourism portal Tiscover,[8] holds detailed information of 14,000 Austrian accommodations, 1,900 Austrian locations, offered leisure activities and relations among them. The Transformation Engine was used to transform this data to RDF conforming to the tourism ontology. This data was fed into a Sesame Triple Store that uses the SwiftOWLIM Sail implementation. The Data Fetchers fetched up to 10 related Web documents per accommodation and location from the World Wide Web using meta-data provided by the Metadata Fetchers. The approximately 150k documents were annotated by the Annotators and indexed by the Indexers to generate the combined index.

To demonstrate HS^3's functionality, consider the query depicted in box 6 of Fig. 8 and replace the keyword "family-friendly" with the keyword "free cancellation". The adapted query is represented as concept description in DL as follows:

$$(\texttt{Hotel} \sqcap \exists \mathit{offers}.(\{\texttt{STEAMBATH}\} \sqcap \texttt{Facility})) \sqcup$$
$$(\texttt{Hotel} \sqcap \exists \mathit{located}.(\{\texttt{MAYRHOFEN}\} \sqcap \texttt{Location}) \sqcup$$
$$(\texttt{Hotel} \sqcap \exists \mathit{describedAs}.(\{\texttt{"FREE CANCELLATION"}\} \sqcap \texttt{Keyword})$$

Therefore, the user is looking for hotels located in Mayrhofen, which are equipped with a steam bath and offer free cancellation. To provide users with additional information and enhance recall the query, expansion mechanism described in Sect. 3.2 is used. In essence the query is expanded to:

$$((\texttt{Hotel} \sqcup \texttt{GuestHouse} \sqcup \texttt{Farm} \sqcup$$
$$\texttt{YouthHostel} \sqcup \texttt{Appartment}) \sqcap$$
$$\exists \mathit{offers}.(\{\texttt{STEAMBATH}\} \sqcap \texttt{Facility})) \sqcup ((\texttt{Hotel} \sqcup \texttt{GuestHouse} \sqcup$$
$$\texttt{Farm} \sqcup \texttt{YouthHostel} \sqcup \texttt{Appartment}) \sqcap$$
$$\exists \mathit{located}.((\{\texttt{MAYRHOFEN}\} \sqcap \texttt{Location}) \sqcup \texttt{Location})) \sqcup ((\texttt{Hotel}$$
$$\sqcup \texttt{GuestHouse} \sqcup \texttt{Farm} \sqcup \texttt{YouthHostel} \sqcup \texttt{Appartment})$$
$$\sqcap \exists \mathit{describedAs}.(\{\texttt{"FREE CANCELLATION"}\} \sqcap \texttt{Keyword}))$$

The result is depicted in Fig. 9. Results are ranked with respect to the user query, as described in Sect. 3.2. Therefore, the top ranked Web sites are those that mention hotels which are equipped with a steam bath, located in Mayrhofen and offer free cancellation. These are followed by Web sites which mention hotels that are equipped with a steam bath and are located in Mayrhofen but do not offer free cancellation, followed by Web sites that mention other accommodations such as guesthouses, farms, apartments that match the criteria and offer free cancellation. Subsequently, Web sites are listed that mention hotels offering free cancellation which are located in Mayrhofen and do not offer a steam bath, followed by hotels not offering free cancellation which are located anywhere and offer a steam bath and so on. Therefore, the results are ranked from the most specific to the least specific.

Every result entry consists of the title of the Web page, a link to the Web page, the first two lines of the summary, a link to the cached page, a link to a list of all concepts and instances

[8] http://www.tiscover.com/.

Fig. 9 Search result

that are mentioned on the page and a drop down element labeled "(– show more –)" that shows the complete summary when expanded. The summary shows all annotations on the page with their context. The annotations are colored in the same color as the corresponding concept in the query, or gray in case the concept is not part of the query. Therefore, the instance "Elisabethhotel" which is mentioned in one of the top ranked documents is colored in the same color as the concept `Hotel` in the query and "Mayrhofen" the same way as `BroadLocation` in the query.

5.1 HS^3 performance evaluation on tourism datasets of different size

To evaluate the performance of HS^3, we created three different datasets based on the tourism KB. HS^3 was instructed to automatically fetch 3,000, 30,000 and 300,000 documents off the World Wide Web. These three different datasets were annotated and a combined index was created. The average term count per document was 1,021 and the average annotation count per document was 90. Furthermore, seven tourism related queries of different complexity were defined. The queries are listed in Table 2.

Every query was executed 1,000 times on the corresponding index and the average execution time was calculated. Neither queries nor results are cached. Every execution takes approximately the same amount of time. We used a commodity notebook with a 2 Gigahertz Dual Core Processor and 4 GB of RAM to run the performance tests. The average execution time in seconds for every dataset and query type is listed in Table 3. It is evident that the keyword only query has the shortest execution time regardless of the document count. Concept

Table 2 Queries for evaluation

Query type	Query
Keyword only	({"ELISABETHHOTEL"} ⊓ Keyword)
Concept simple	Hotel
Concept simple & keyword	Hotel ⊓∃ describedAs.({"CHILD-FRIENDLY"} ⊓ Keyword)
Concept standard	Hotel ⊓∃ locatedIn.({MAYRHOFEN} ⊓ Location)
Concept standard & keyword	(Hotel ⊓∃ locatedIn.({MAYRHOFEN} ⊓ Location)) ⊔ (Hotel ⊓∃ describedAs.({"CHILD-FRIENDLY"} ⊓ Keyword))
Concept complex	(Hotel ⊓∃ locatedIn.({MAYRHOFEN} ⊓ Location)) ⊔ (Hotel ⊓∃ offers.({STEAMBATH} ⊓ Facility))
Concept complex & keyword	(Hotel ⊓∃ locatedIn.({MAYRHOFEN} ⊓ Location)) ⊔ (Hotel ⊓∃ offers.({STEAMBATH} ⊓ Facility)) ⊔ (Hotel ⊓ describedAs.({"CHILD-FRIENDLY"} ⊓ Keyword))

Table 3 HS^3 performance in seconds per index document count and query type

Query \Doc count	3K	30K	300K
Keyword only	0.0003	0.001	0.006
Concept simple	0.028	0.105	0.909
Concept simple & keyword	0.036	0.131	1.046
Concept standard	0.030	0.100	0.826
Concept standard & keyword	0.052	0.126	0.929
Concept complex	0.046	0.118	0.857
Concept complex & keyword	0.066	0.153	0.995

only search queries are more complex than keyword only queries and result in higher query times. However, they still are in the range of 1 s for big datasets.

Furthermore, it can be observed that the execution time increases only marginal between a simple concept and complex concept query no matter how many documents have been indexed. This also holds true for the combined search, including keywords and concepts. Compared to the execution times of the concept only search queries, the execution times of the combined queries only increase marginally. (Due to a bug in the scoring mechanism of SIREn, we had to use a workaround which resulted in more complex queries and higher execution times. Hence, lower execution times can be realized when the bug gets fixed.)

5.2 Precision and recall evaluation

To conduct an unbiased precision and recall evaluation for semantic search systems and hybrid search systems, publicly available test-datasets are needed. Precision and recall of traditional IR systems can be evaluated by using the widely accepted TREC datasets,[9] which includes a document collection, a set of queries and judgments representing the ground truth. However, the TREC datasets are tailored for the evaluation of traditional IR systems but lack a publicly available ontology, KB as well as related ground truth data. To compare

[9] http://trec.nist.gov/.

different semantic systems accurately in terms of precision and recall all systems need to operate on the same document corpus, the same ontology and KB, use the same queries and need to have the same ground truth data. As datasets that fulfill these requirements are not available for semantic search systems, we used the publicly available Harmonise ontology and KIM Ontology to build a document corpus for each domain by using HS^3's automatic document fetching functionality. The Harmonise ontology is settled in the e-Tourism domain and the KIM Ontology in the news domain. HS^3 fetched and annotated about 167,029 unique documents for the KIM Ontology and 1,698 unique documents for the Harmonise ontology. We intentionally decided to generate a big corpus, the KIM document corpus, and a small corpus, the Harmonise document corpus, to be able to assess the scalability of the system as well. Furthermore, we wanted one corpus, the Harmonise document corpus, where it is possible to manually asses the ground truth by having a manageable amount of documents. Due to the high document count of the KIM document corpus, we had to use semi-automatic approaches to generate the ground truth set, which results in a less accurate ground truth compared to the manually created one of the Harmonise document corpus. We defined 18 queries for the Harmonise and KIM corpus in total. Queries that are similar to common user queries which have been issued on a tourism portal were used for the quality evaluation on the Harmonise document corpus and queries that are similar to the queries used by Castells et al. [7] on a similar news corpus were used for the quality evaluation on the news document corpus. The data and query sets used in the evaluations are available from the project homepage[10] to serve as benchmark for further evaluations.

In the evaluation, a document qualifies as relevant if it contains information that satisfies the information need of the user formulated in the search query. The information can either be available in the document as text or via an annotation that is linked to supplementary information in the Knowledge Base. For example, consider the query "hotel located in mayrhofen with a steam bath". For this case, documents that contain textual descriptions of hotels which offer a steam bath and are located in Mayrhofen are relevant but also those documents that contain an annotation of type `Hotel` which links to a corresponding entry in the Knowledge Base that holds the relevant information are deemed as relevant. Therefore, documents are also deemed relevant if the annotation points to an entry in the Knowledge Base that holds the information that the hotel is located in Mayrhofen even though the term Mayrhofen is not present in the document.

We evaluated precision and recall for the Harmonise and KIM corpus datasets and calculated a precision and recall curve for each set. The average precision and recall curve of the e-Tourism (Harmonise) dataset (Fig. 10a) and the news dataset (Fig. 10b) shows that the hybrid semantic search approach is in general superior to the keyword-based approach. The high precision of the hybrid search approach on the e-Tourism dataset results from the combined search approach and the sophisticated annotation and extraction rules that have been specifically tailored to the tourism domain. It can be seen that the precision gets lower at higher recall levels for the news dataset. The reason for this is that the annotation rules that have been used for the news dataset are more general than the ones used for the e-Tourism dataset, resulting in a number of wrong and missed annotations, leading to lower precision at higher recall levels. The average precision and recall curve calculated from the precision and recall curves of the respective sets is depicted in Fig. 10c.

It can be seen from the average precision and recall curve that the hybrid approach's precision in general is higher on any recall level than the precision of the keyword only approach. Furthermore, even if the semantic part of the hybrid approach cannot contribute

[10] http://www.ifs.tuwien.ac.at/ir/hybridsearch.

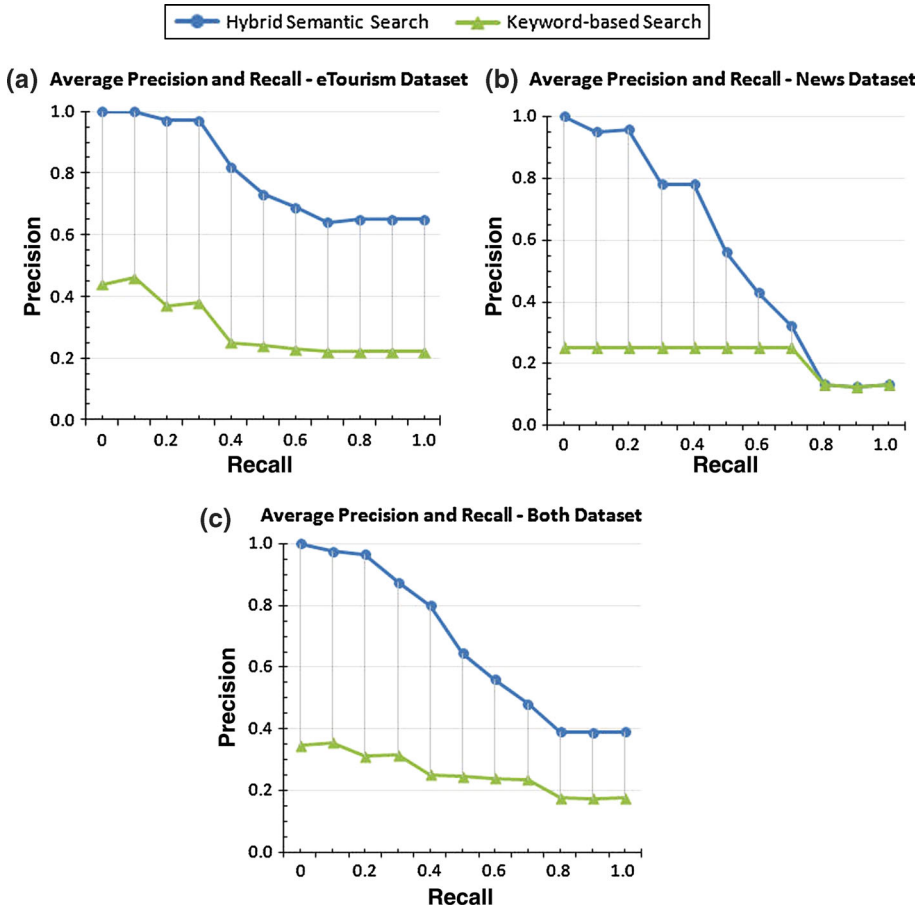

Fig. 10 Results of the precision and recall evaluation

anything to the search result, due to missing annotations or a lack of suitable concepts in the ontology, it still can make use of the keyword-based approach to deliver the same results as the keyword only search approach. The system has high precision rates on low recall levels but still maintains a good precision as the recall level increases. What distinguishes HS^3 from other systems with similar capabilities is that HS^3 needs at most 0.618 s and on average only 0.218 s to return the result for a complex query on the corpus of 167,029 annotated news documents.

5.3 How to apply HS^3 to a different domain

HS^3 has been designed to work with an arbitrary OWL-based ontology and KB. In this section we discuss the steps that need to be taken to apply HS^3 to a different domain. In case the initial knowledge is not available in a data format suitable for loading it into the Triple Store, it needs to be transformed with the Transformation Engine. The Transformation Engine has a RDBMS and text / HTML file plug-in. If a different type of data source is used, a custom plug-in needs to be implemented. For the Transformation Engine to transform a custom data format into

structured data, a transformation rule set needs to be defined. Therefore, the prerequisite is that the initial data is either already stored in the KB or loaded into the KB via the Transformation Engine. Next, the Knowledge Engineer needs to define the *Realms* (see Sect. 3.2 for details) and the *Index Graphs* (see Sect. 3.1 for details) with respect to the target audiences' information need and the used ontology. Next, the Annotation component and Concept & Instance Suggestor need to be equipped with textual descriptions of concepts and instances. Descriptions can be either attached via the properties `hs3:prefTextualRepresentation` and `hs3:altTextualRepresentation` or by the default `rdfs:label` property to concepts and instances.

When all prerequisites are met, HS^3 can be started. On the first start up the initial suggestion index for the Concept & Instance Suggestor is build automatically. Subsequently, Metadata Fetchers and Data Fetchers are started and fetch related data off the World Wide Web and put it as work packages into the work queues. Annotators pick up the work packages, annotate the included Web pages via their annotation pipelines and put the work packages back into the work queue. Indexers pick up these work packages, analyze the annotations of the documents and update the combined index. In parallel HS^3 can already process user search requests by using the current combined index. All Fetchers, Annotators and Indexer work iteratively to keep the system's information up to date. To showcase how to apply HS^3 to a different domain we used the KIM Ontology [25] (KIMO) and HS^3's standard mechanisms to create the suggestion index. A minor extension of the standard mechanism was needed, because the KIM Ontology uses the properties `hasMainAlias` and `hasAlias` to refer to the textual representations of instances and concepts. To demonstrate the applicability of HS^3 for arbitrary ontologies and determine the user acceptance of the interactive ontology-enhanced keyword-based input mechanism we conducted a user evaluation. The evaluation was based on the tourism ontology and on the KIM ontology.

Even though it is possible to use the system with an arbitrary ontology certain restrictions have to be considered. First, the bigger the ontology is the bigger the index gets if the *Index Graphs* are defined very broadly. For example, especially the medical domain is very broad and to use the entire domain might not be manageable or cause potential performance issues. For huge ontologies such as medical ontologies the index can get big in case every concept is equipped with an *Index Graph* and documents contain a lot of annotations. Therefore, it is possible to limit the number of instances that are indexed by the Indexer by providing it with the maximal instance count per document that should be indexed. In this case the Indexer indexes the N most relevant instances that are mentioned in a document. The relevance of an instance is determined by checking whether that instance is mentioned in the title of the document, if related instances that are part of the instance's *Index Graph* also appear in the document (e.g. the location in case of a hotel) and how often the instance is mentioned in the document. Second, some ontologies are structured badly which makes it hard to define accurate *Index Graphs*. For example an ontology which has a root concept and all other concepts of the ontology are direct sub-concepts of the root concept is badly structured making it difficult to define accurate *Index Graphs*.

5.4 User evaluation of the interactive ontology-enhanced keyword-based input mechanism

The purpose of the evaluation was to determine whether average Internet users are able to efficiently formulate semantic queries, without prior knowledge of the ontology with the interactive ontology-enhanced keyword-based input mechanism. We structured our evaluation into 5 sections. In the first section, participants were asked to complete a Pre-Questionnaire, which was used to determine their Internet and search engine usage and to test their knowledge

Table 4 Queries

Query no.	Query
1.	APARTMENT
2.	HOTEL ⊓∃ locatedIn.“Mayrhofen”
3.	(FARM ⊓∃ locatedIn.BROADLOCATION) ⊔ (BROADLOCATION ⊓ ∃ languageSpoken.ENGLISH)
4.	(HOTEL ⊓∃ locatedIn.“Retz”) ⊔ (HOTEL ⊓ ∃ offers.TELEPHONESERVICE)
5.	(FARM ⊓∃ providesFacility.GARDEN) ⊔ (FARM ⊓∃ languageSpoken.ENGLISH) ⊔ (FARM ⊓∃ describedAs.({“QUIET”} ⊓ Keyword))
6.	PERSON ⊓∃ describedAs.({“IMPORTANT”} ⊓ Keyword)
7.	(PERSON ⊓ ∃ holdsPosition.CEO) ⊔ (CEO ⊓ ∃ ofOrganization.“Apple”)
8.	(BANK ⊓ ∃ isOwnedBy.PERSON) ⊔ (PERSON ⊓ ∃ hasRelative.“Donald Harris”)
9.	(COMPANY ⊓ ∃ isParentOrganizationOf.“Cisco Systems”) ⊔ (COMPANY ⊓ ∃ partiallyControls.“Neogen”)

about the Semantic Web in general. This section was followed by a short description of the differences among concepts, instances of concepts and keywords in the context of the Semantic Web. Furthermore, a short introduction on how to use the interface was given by means of an example. The two main sections asked the participants to formulate different search queries, which were formulated in natural language, with the interactive ontology-enhanced keyword-based input mechanism. The first main section contained five queries from the tourism domain based on the tourism ontology, whereas the second main section held five queries from the news and broadcast domain based on the KIM Ontology. The last question asked the users to formulate an arbitrary search query that involves an arbitrary number of concepts and instances of the concept `Person` and `Organization` or its sub-concepts. This was followed by a text box where users were asked to formulate their search intend in natural language to determine whether a user's search intend was reflected in the constructed query. In the last section participants were asked to complete a Post-Questionnaire, which was used to determine their general attitude toward this new form of search query creation. Table 4 depicts the queries users were asked to formulate and Table 5 lists the percentage of users that rated the query formulation either very easy, easy, OK, hard or very hard.

Twenty-two test persons participated in the user evaluation. They were between 22 and 52 years old and moderate to regular Internet users. About 73 % were male and 27 % were female. They worked either in consulting, customer service, health care, management, logistics, research, software development or were enrolled as students at an university. All except two participants stated Google as their preferred search engine. Furthermore, the majority of test persons stated that they use on average between 3 and 4 terms for their search queries. About 77 % of the participants have heard of the Semantic Web and about 59 % knew what ontologies are. However, more than half of all participants did not know the difference between concept, instance of concept and keyword. Still, the majority of test persons were able to use the

Table 5 Perceived difficulty of query formulation

Query no.	Ontology	Very easy (%)	Easy (%)	OK (%)	Hard (%)	Very hard (%)
1.	e-Tourism	**77.3**	18.2	0	4.5	0
2.	e-Tourism	**54.5**	36.4	9.1	0	0
3.	e-Tourism	27.3	**59.1**	13.6	0	0
4.	e-Tourism	27.3	**36.4**	31.8	4.5	0
5.	e-Tourism	9.1	22.7	**40.9**	18.2	9.1
6.	KIM	**31.8**	13.6	22.7	22.7	9.1
7.	KIM	22.7	**40.9**	31.8	4.6	0
8.	KIM	22.7	**45.5**	27.3	4.6	0
9.	KIM	22.7	**36.4**	31.8	9.1	0

interface intuitively without knowing the difference. It can be seen from Table 5 that the majority of participants did not have any difficulties formulating the search queries with the interface and stated that query formulation was either very easy, easy or OK. On average the participants got 8.227 queries out of 9 queries correct with a standard deviation of 0.922. It is noticeable that those queries, namely 5 and 6, which asked the user to use a keyword instead of an instance or concept to extend the query have been perceived as hard or very hard by the minority of test persons. Only 14 participants out of 22 formulated query 5 correctly, even though it is a rather simple query, and about 18 participants out of 22 formulated query 6 correctly. We think that this relates to the problem that not all participants understood the difference between a keyword or term on a Web page and an annotation that refers to a concept, instance or both in the ontology and KB. Hence, the actual meaning of keyword, concept and instance in the context of the Semantic Web and World Wide Web should have been explained more comprehensively, to provide these participants with the big picture.

However, when asked about the difficulty to formulate queries with the interface in general, about 9.1 % of users stated that it is very easy, about 40.9 % stated that it is easy and about 50 % that it is OK. The last question, which asked the participants to formulate an arbitrary search query with the given ontology, showed that people had a good understanding of the ontology even though they had no prior knowledge of it. Most of the participants formulated a rather complex query involving different concepts, instances, properties and keywords. To check whether they actually understood what they had formulated, we asked them to formulate the same query in natural language. Various test persons stated that they liked the clean and tidy interface as well as the fast retrieval of suggestions. Furthermore, the participants stated that the different colors of the concepts in the graphical query were helpful during the query generation. All test persons stated that the concept and property suggestion mechanism was either helpful or very helpful. A few participants criticized that it was difficult to edit the graphical query once defined, because the editing box did not appear instantly or not at all. This issue is related to the used Internet Browser and will be fixed in a future release. Additional information is available at the project homepage.[11]

6 Conclusion

We are still far from creating large-scale hybrid search engines, compared to large-scale keyword-based search engines such as the ones of Google, Yahoo or Microsoft that can

[11] http://www.ifs.tuwien.ac.at/ir/hybridsearch.

bridge the gap between structured and unstructured data completely and efficiently. The main challenge remains the creation of big combined indexes that encode the knowledge of the World Wide Web and the Semantic Web. As combined indexes and the techniques to create them are more complex than the techniques and creation of "simple" indexes, which are currently used by keyword-based search engines, considerable computing power is needed to create them. For now the biggest combined index has been create for ESTER, combining semantic knowledge from the YAGO ontology and KB and textual data from Wikipedia. However, to further advance the creation and user acceptance of hybrid search systems, we introduced a novel input mechanism for hybrid semantic search that combines the clean and concise input mechanisms of keyword-based search engines with the expressiveness of the input mechanisms provided by semantic search engines. The interface amalgamates concept-based and keyword-based input. By using this interface, users can formulate queries without prior knowledge of the underlying ontology. We proposed an architecture than can be used to automatically fetch relevant unstructured information from the World Wide Web to complement structured information in a KB. The architecture can be used to automatically fetch, annotate, and index unstructured data to create a combined index that amalgamates unstructured and structured information. Furthermore, the structure of the combined index has been presented.

We introduced the Hybrid Semantic Search System HS^3 which implements the proposed architecture, combined index, and novel input mechanism. We gave an overview of HS^3's architecture, its components and an in-detail presentation of the Search & Ranking components. We applied HS^3 to the tourism domain and presented a showcase and performance test results. We discussed *Index Graphs*, which are used to encode information of ontology instances, that can be used during the retrieval process. We used keyword-based, concept-based, and hybrid queries of different complexity to evaluate HS^3's performance. The performance evaluation showed that response times of the system are within a second even for big datasets and complex queries consisting of a combination of concepts, instances and keywords. Furthermore, the performance tests showed that only a marginal difference between simple queries and complex queries in terms of response time exists.

The interactive ontology-enhanced keyword-based input mechanism has been showcased using the tourism dataset and the KIM Ontology and KB. To evaluate the usability of the novel input mechanism we conducted a user evaluation to ascertain the acceptance of the input mechanism, whether it is easy or hard to formulate queries of different complexity and whether it is possible to formulate queries without prior knowledge of the ontology. The user evaluation indicated that average Internet users are able to use the interface without any difficulties and that they deem the concept, instance and property suggestion mechanism as very helpful. The test persons liked the clean and tidy interface as well as the fast retrieval of suggestions. Furthermore, the evaluation showed that the participants were able to formulate expressive queries involving concepts, instances, properties, and keywords without prior knowledge of the ontology.

Regarding future work, we plan to enhance the interactive ontology-enhanced keyword-based input mechanism with a context-aware suggestion mechanism that incorporates the query context into the suggestion mechanism. Furthermore, we plan to simplify the editing mechanism and provide users with the possibility to annotate their input with concepts of the ontology. Finally, we plan to demonstrate how to integrate HS^3 via Web Services and agents into a virtual environment by integrating it into the 3D e-Tourism Environment Itchy-Feet [13] via HS^3's Web Service interface.

References

1. Baader F, McGuinness D, Nardi D, Patel-Schneider P (2003) The description logic handbook: theory, implementation and applications. Cambridge University Press, Cambridge
2. Bast H, Bäurle F, Buchhold B, Haussmann E (2012) Broccoli: semantic full-text search at your fingertips. CoRR abs/1207.2615
3. Bast H, Chitea A, Suchanek F, Weber I (2007) Ester: efficient search on text, entities, and relations. In: Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval, SIGIR'07. ACM, New York, NY, USA, pp 671–678
4. Bhagdev R, Chapman S, Ciravegna F, Lanfranchi V, Petrelli D (2008) Hybrid search: effectively combining keywords and semantic searches. In: Proceedings of the 5th European semantic web conference on The semantic web: research and applications, ESWC'08. Springer, Berlin, Heidelberg, pp 554–568
5. Bikakis N, Giannopoulos G, Dalamagas T, Sellis T (2010) Integrating keywords and semantics on document annotation and search. In: Proceedings of the 2010 international conference on the move to meaningful internet systems: Part II, OTM'10. Springer, Berlin, Heidelberg, pp 921–938
6. Broekstra J, Kampman A (2003) Serql: a second generation RDF query language. In: Proceedings of the 2003 SWAD-Europe workshop on semantic web storage and retrieval
7. Castells P, Fernandez M, Vallet D (2007) An adaptation of the vector-space model for ontology-based information retrieval. IEEE Trans Knowl Data Eng 19:261–272
8. Cunningham H, Maynard D, Bontcheva K, Tablan V (2002) Gate: an architecture for development of robust hlt applications. In: Proceedings of the 40th annual meeting on association for computational linguistics, ACL'02. Association for, Computational Linguistics, pp 168–175
9. Delbru R, Toupikov N, Catasta M, Tummarello G (2010) A node indexing scheme for web entity retrieval. In: Proceedings of the extended semantic web conference (ESWC 2010), pp 240–256
10. Fernandez M, Lopez V, Sabou M, Uren V, Vallet D, Motta E, Castells P (2008) Semantic search meets the web. In: Proceedings of the 2008 IEEE international conference on semantic computing. IEEE Computer Society, Washington, DC, USA, pp 253–260
11. Fodeh S, Punch B, Tan PN (2011) On ontology-driven document clustering using core semantic features. Knowl Inf Syst 28:395–421
12. Fodor O, Werthner H (2005) Harmonise: a step toward an interoperable e-tourism marketplace. Int J Electron Commer 9:11–39
13. Gärtner M, Seidel I, Froschauer J, Berger H (2010) The formation of virtual organizations by means of electronic institutions in a 3d e-tourism environment. Inf Sci 180:3157–3169
14. Giunchiglia F, Kharkevich U, Zaihrayeu I (2009) Concept search. In: Proceedings of the 6th European semantic web conference on the semantic web: research and applications, ESWC 2009 Heraklion. Springer, Berlin, Heidelberg, pp 429–444
15. Guha R, McCool R, Miller E (2003) Semantic search. In: Proceedings of the 12th international conference on World Wide Web, WWW'03. ACM, New York, NY, USA, pp 700–709
16. Hearst MA (2009) Search user interfaces, 1st edn. Cambridge University Press, New York
17. Jalali V, Matash Borujerdi M (2011) Information retrieval with concept-based pseudo-relevance feedback in medline. Knowl Inf Syst 29:237–248
18. Kandogan E, Krishnamurthy R, Raghavan S, Vaithyanathan S, Zhu H (2006) Avatar semantic search: a database approach to information retrieval. In: Proceedings of the 2006 ACM SIGMOD international conference on Management of data, SIGMOD'06. ACM, New York, NY, USA, pp 790–792
19. Kasneci G, Suchanek FM, Ifrim G, Elbassuoni S, Ramanath M, Weikum G (2008) Naga: harvesting, searching and ranking knowledge. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, SIGMOD'08. ACM, New York, NY, USA, pp 1285–1288
20. Kaufmann E, Bernstein A (2010) Evaluating the usability of natural language query languages and interfaces to semantic web knowledge bases. J Web Semant 8:377–393
21. Kaufmann E, Bernstein A (2007) How useful are natural language interfaces to the semantic web for casual end-users? In: Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference, ISWC'07/ASWC'07. Springer, Berlin, Heidelberg, pp 281–294
22. Kiryakov A, Popov B, Ognyanoff D, Manov D, Goranov KM (2004) Semantic annotation, indexing, and retrieval. J Web Semant 2:49–79
23. Mukherjea S (2004) Discovering and analyzing world wide web collections. Knowl Inf Syst 6:230–241
24. Navarro G (2001) A guided tour to approximate string matching. ACM Comput Surv 33:31–88
25. Popov B, Kiryakov A, Ognyanoff D, Manov D, Kirilov A (2004) Kim a semantic platform for information extraction and retrieval. J Nat Lang Eng 10:375–392
26. Prud'hommeaux E, Seaborne A (2004) Sparql query language for RDF. Technical report W3C

27. Reeve L, Han H (2005) Survey of semantic annotation platforms. In: Proceedings of the 2005 ACM symposium on applied computing, SAC'05. ACM, New York, NY, USA pp 1634–1638
28. Salton G, McGill MJ (1986) Introduction to modern information retrieval. McGraw-Hill, Inc., New York
29. Schreiber G, Amin A, Aroyo L, van Assem M, de Boer V, Hardman L, Hildebrand M, Omelayenko B, van Osenbruggen J, Tordai A, Wielemaker J, Wielinga B (2008) Semantic annotation and search of cultural-heritage collections: the multimedian e-culture demonstrator. J Web Semant 6:243–249
30. Snchez D, Isern D, Millan M (2011) Content annotation for the semantic web an automatic web-based approach. Knowl Inf Syst 27:393–418
31. Stumme G, Hotho A, Berendt B (2006) Semantic web mining: state of the art and future directions. J Web Semant 4(2):124–143
32. Suchanek F, Kasneci G, Weikum G (2008) YAGO: a large ontology from Wikipedia and WordNet. J Web Semant 6(3):203–217
33. Tablan V, Damljanovic D, Bontcheva K (2008) A natural language query interface to structured information. In: Proceedings of the 5th European semantic web conference on the semantic web: research and applications, ESWC'08, Springer, pp 361–375
34. Wang H, Tran T, Liu C (2008) Ce2: towards a large scale hybrid search engine with integrated ranking support. In: Proceeding of the 17th ACM conference on Information and knowledge management, CIKM '08. ACM, New York, NY, USA, pp 1323–1324
35. Zenz G, Zhou X, Minack E, Siberski W, Nejdl W (2009) From keywords to semantic queries-incremental query construction on the semantic web. J Web Semant 7:166–176
36. Zha ZJ, Yang L, Mei T, Wang M, Wang Z (2009) Visual query suggestion. In: Proceedings of the 17th ACM international conference on multimedia, MM'09. ACM, New York, NY, USA, pp 15–24

Author Biographies

Markus Gärtner is currently working as Solution Designer and Project Manager at the telecommunications company T-Mobile Austria. Starting August 2013, he will pursue an MBA study at the London Business School. He received his M.Sc. and Ph.D. in Computer Science from the Vienna University of Technology in 2006 and 2012, respectively. Markus Gärtner developed a multi-agent e-Commerce Trading Environment based on Electronic Institutions, a Multi-Agent System paradigm, as part of his work for an FWF funded project. Furthermore, he designed and developed the prototype of a Hybrid Semantic Search System. His research interests include Multi-Agent and Distributed Systems, agent technology, information retrieval, Semantic Web and Search, security, project management, software engineering as well as Virtual and Augmented Reality.

Andreas Rauber is Associate Professor at the Department of Software Technology and Interactive Systems (ifs) at the Vienna University of Technology (TU-Wien). He furthermore is president of AARIT, the Austrian Association for Research in IT. He received his M.Sc. and Ph.D. in Computer Science from the Vienna University of Technology in 1997 and 2000, respectively. In 2001, he joined the National Research Council of Italy (CNR) in Pisa as an ERCIM Research Fellow, followed by an ERCIM Research position at the French National Institute for Research in Computer Science and Control (INRIA), at Rocquencourt, France, in 2002. His research interests cover the broad scope of digital libraries and information spaces, including specifically text and music information retrieval and organization, information visualization, as well as data analysis, neural computation and digital preservation.

Helmut Berger was head of the Information Retrieval department of an Austrian service provider for solutions in the patent and intellectual property domain. During this time, he gained extensive experience in the areas of Text Analytics and Information Retrieval with particular emphasis on the challenges in the patent domain. Before that, he was Senior Researcher at an Austrian center for applied research in the area of e-Commerce. His work concentrates on patent information retrieval, data mining and machine learning, and human–computer interaction. In these areas, he has published more than 60 papers in books, conference proceedings, and journals. He has studied Computer Science at the Vienna University of Technology. Helmut received his master's degree in Computer Science in 2001 and his PhD in 2003 from the Vienna University of Technology.