Squiggle: a Semantic Search Engine for indexing and retrieval of multimedia content

Irene Celino, Emanuele Della Valle, Dario Cerizza, and Andrea Turati

CEFRIEL – Politecnico of Milano, Via Fucini 2, 20133 Milano, Italy
{irene.celino,emanuele.dellavalle,dario.cerizza,andrea.turati}@cefriel.it

Abstract. Search engines are becoming such an easy way to find textual resources that we wish to use them also for multimedia content; however, syntactic techniques, even if promising, are not up to the task: future search engines must consider new approaches. Experimental prototypes of this search engine of the future are appearing. Most of them employs "smart machines" able to directly elaborate multimedia resources, but we believe that the solution should embrace also "smart data", able to capture lexical and conceptual characteristics of a domain in an ontology. In order to prove that Semantic Web technologies provide real benefits to end users in terms of an easier and more effective access to information, we developed Squiggle, a Semantic Web framework that eases the deployment of semantic search engines. Following a model-driven approach to application development, Squiggle makes ontologies (both the SKOS model and the domain knowledge) part of the running code. We evaluate the advantages of Squiggle against traditional approaches in two real world deployments: one to search images of skiers for Torino 2006 Winter Olympic Games and one to search music files.

1 Introduction

Finding resources in loosely structured repositories (e.g. the Web, a file server, even our own mailboxes), in which users can freely add more data, is one of the key problems of the "digital era". In most cases, such problem is addressed by offering a search engine that periodically crawls resources, indexes them and enables fast searches over those indexes.

Searching everything everywhere is becoming our habit when we need to find something. We search Web pages in Web search engines, music using search engines integrated in multimedia players, pictures in images organizer applications, movies using systems such as Blinkx.tv, even personal stuff using desktop searches.

However, *finding* what we need is often a hard job. Current search engine technology [1] is very good in finding complete Web pages published all over the world, but it lacks the desired precision[1] and recall[2] when searching for multimedia resources. For instance, searching "jaguar" in an image search engine

[1] Precision is the proportion of relevant data of all data retrieved.
[2] Recall is the proportion of retrieved relevant data, out of all available relevant data.

(e.g., Google, Yahoo, MSN, etc.) results in a mix of felines and cars, which are difficult to tell apart. Moreover, current technology is unable to cope with results that requires either to extract a part of a resource (e.g., a scene from a movie) or to aggregate numerous resources (e.g., relevant but scattered information regarding a person).

Furthermore, searching is an *expensive activity*. For instance, in a medium-sized enterprise with 100 employees, each of them would perform around 10 searches per day (some on the Web, some on their mailboxes, etc.), stopping, successfully or not, in 1-2 minutes. This means that 20-30 hours a day are spent in searching. However, loosing time in searching is not the only source of costs, since getting lost in results and missing relevant information can lead to loose important opportunities. We may not care about the efficiency loss for an enterprise, but, as citizens, we care about saving our own time: for example, if we cannot find a document in an e-government website, we are forced to go personally to a counter.

What we really need is a search engine able to find any kind of multimedia resources with the required level of granularity; but, how can we achieve this *search engine of the future*? We believe that Tim Berners-Lee was right when, drawing the "Semantic Web Roadmap"[2], he said:

> If an engine of the future combines a reasoning engine with a search engine, it may be able to get the best of both worlds.

Keeping in mind Tim Berners-Lee claim, we conceived, implemented and deployed Squiggle[3] (*S*emantically *qui*lted *goo*gle), an extensible semantic search framework designed to add a conceptual flavour at indexing time and to exploit as much as possible ontological elements to improve searching time.

The paper is organized as follows. Section 2 outlines the current efforts for extending syntactic search combining smart data (i.e., metadata defined by ontologies using Semantic Web standards) and smart machines (e.g., image processing). The approach we foresee for the "search engine of the future" follows in section 3. In section 4 it is shown the conceptual architecture of the Squiggle framework. Real world deployment of Squiggle, which demonstrate the feasibility of our approach, are analyzed in section 5. Finally, concluding remarks are provided in section 7.

2 Existing approaches to improve search engines

Syntactic search techniques are largely employed not only on the Web, but also in many applications we daily use for our productivity or pleasure. A standard "syntactic" search engine's implementation is mainly based on three phases (cf., among others, [1]):

- crawling time: it is the phase in which the resources (HTML pages, multimedia contents, etc.) are collected in order to build a coherent (and more homogeneous) set;

[3] Visit Squiggle website at: `http://squiggle.cefriel.it`

- indexing time: it is the phase during which the crawled resources are parsed and indexed in some particular data structures; those structures are built based on the relevant information contained in indexed resources and are optimized to quickly answer to queries (granting search response-time in the order of milliseconds);
- searching time: it is the run-time phase in which final users submit their queries in order to retrieve meaningful results; in addition to the optimization of the indexes, this phase requires also a good method to rank and/or cluster search results.

The reasons why Web search engines give such good results, in terms of performance and scalability, relies in the very form of the Web, made up of *semi-structured text* and *links*. At indexing time, Web pages, being semi-structured text, can be subject to a wide range of well known techniques for parsing tags, tokenising contents of tags, and building indexes. Moreover, Web resources, being interlinked, enable a straightforward implementation of spiders, which can traverse the entire Web at crawling time, and the application of effective ranking algorithms (e.g., PageRank [3]) useful at search time.

When the resources to be indexed are multimedia files instead of Web pages, the automatic process of their content becomes very difficult and the lack of links makes crawling a tricky problem and PageRank algorithm useless. The two ways out are the use of smart machines and smart data.

By *smart machine* we mean a bunch of techniques that includes text processing (e.g., natural language processing), audio processing (e.g. acoustic fingerprints, automatic speech recognition, etc.) and image/video processing (e.g. computer vision, scene change detection, segment detection, etc.). Several search engines that exploit smart machines are appearing. For instance, Retrievr[4] finds images by drawing rough sketches of them; Musipedia[5] offers an interface for querying by humming; ANSES[6] (Automatic News Summarization and Extraction System) captures TV news, extracts key scenes from the video, analyzes the audio extracting references to key persons, places, date-time and enables searches on the repository.

On the other side, *smart data* is the base for search engines that exploits semantics at search time to increase both recall and precision, with respect to engines that are purely syntactic that do not exploit the possibility, offered by Semantic Web standards, of modeling the domain both at lexical (i.e., SKOS [4]) and at knowledge level (i.e., OWL [5]). Explicit representation of semantics (both lexicon and knowledge), gives search engines the ability to disambiguate between homonyms and expand the search to synonyms and pseudonyms. Moreover, smart data gives search engines the possibility of augmenting the set of relevant results by exploiting: hyperonymy and hyponymy at lexical level, conceptual broadening and narrowing at knowledge level or any other relationship

[4] http://labs.systemone.at/retrievr
[5] http://www.musipedia.org/
[6] http://www.doc.ic.ac.uk/∼mjp3/anses/

that can be described by a domain ontology (e.g., "The Three Tenors" is the brand used by Pavarotti, Domingo and Carreras when singing together).

However, at search time all these features are available only if resources are augmented with semantic annotations, which don't come for free. The most obvious way to semantically annotate resources is doing it manually (for instance using Annotea [6] or SMORE [7]). Clearly, such manual process is affordable only in specific domains in which either cultural reasons (e.g., the librarians have been annotating and cataloging books since ever) or collaborative behaviors (e.g., Wikipedia) make the annotation process sustainable. In all other cases, some (semi)automatic annotation mechanisms is needed. To this end, combining smart data with smart machine seems to be the right approach.

There are several examples of existing approaches that try to *combine Semantic Web technologies with smart machines* in search engines. One of the most interesting is represented by KIM [8]. KIM starts from the idea that a practical semantic annotation is impossible without some particular knowledge modeling commitments. Therefore, KIM introduces a light-weight upper-level ontology, which encodes many of the domain-independent commonsense concepts and allows domain-specific extensions. Moreover, KIM includes a semantically enhanced information extraction system, which provides automatic semantic annotation with references to classes and instances in the ontology. Finally, on the basis of these semantic annotations, KIM performs semantic based indexing and retrieval, where users can mix traditional queries and ontology-based ones.

Some other interesting approaches in introducing semantics in indexing and searching are:

- TAP [9] Semantic Search, which has two goals: (1) augmenting traditional search results with data pulled from the Semantic Web; (2) using an understanding of the "*denotation*" of the search term to improve traditional search, where denotation is the problem of identifying the concepts in the search query;
- ALVIS [10] open-source semantic-based peer-to-peer search engine, which, before indexing the documents, enriches them with some "semantic awareness" of the specific subject, by using information extraction technology;
- ConWeaver [11] semantic search engine, which uses a knowledge network (i.e., an ontology), as a rich knowledge-based search index, thus enabling the restitution of search results for queries in multilingual, heterogeneous data environments.

3 Our steps towards the "search engine of the future"

In our opinion, what Tim Berners-Lee calls the "search engine of the future" should have a structure similar to existing "syntactic" search engines, but should also be enriched with machine-processable semantics. In our vision, domain ontologies can be employed in empowering searching, indexing and also crawling.

At *crawling time*, a previous knowledge about the domain can assist the collecting of resources to be indexed, because this "know-how" can drive the

crawler to focus on relevant information even if links are not explicit. For example, the awareness about a specific domain knowledge (e.g., the "Beatles" used to be called the "fab four" and that they are frequently misspelled "Beetles") promotes and eases the individuation of the documents containing relevant information (e.g., music of the Beatles within a folder full of mp3 files that normally include in the filename the indication of the performer, in its multiple spellings).

At *indexing time*, "a little semantics" can be introduced: the input information can be analyzed by means of smart machines and tagged with respect to its meaning before it is processed by the indexer tool. In this way, the tool is able to index both the syntactic content of an input document and its attached semantics (i.e., the unique identifiers of the concepts that tag the document). In brief, the indexing process can be driven by semantics, becoming an effective conceptual indexing process.

At *searching time*, domain ontologies can be employed to customize search engine applications and to improve the user experience in terms of value added and effectiveness of the search. If aware of specific concepts and properties, the tool can help the user in refining his query both by clarifying the matter of his search ("Did you mean...?") and by suggesting possible expansions of his query to related subjects ("You could also be interested in..."). The result is that the user can find more easily what he was looking for.

We conceived Squiggle, keeping in mind Tim Berners-Lee claim and the above analysis. In particular, Squiggle is an extensible framework (see §4) designed to add a conceptual layer to indexing process and to exploit as much as possible ontological elements to improve searching time, leaving to each domain-dependent instantiation of the framework the choice of using ontologies also at crawling time (see Squiggle case studies in §5).

4 Conceptual architecture of the Squiggle framework

As a result Squiggle is :

- a *semantic search-engine*, i.e. a semantic web application with searching functionalities; and
- a *semantic-search engine*, i.e. a search engine that is able to deal with the "meaning" of the searched information.

While the latter objective concerns the semantics of the *data* and can be achieved through an opportune modeling of knowledge, the former purpose is strictly related to the model-driven development of a semantic web *application*.

Squiggle is indeed a semantic application, since its design heavily grounds on a common model, provided by SKOS [4], which permeates all its structure; Squiggle assumes SKOS as its application ontology and, therefore, is naturally able to exploit SKOS' semantics (mainly its properties, like `prefLabel/altLabel`, `broader/narrower`, `related/relatedPartOf`, etc.).

Moreover, being SKOS a horizontal ontology useful to capture schemata of concepts (therefore not bound to any specific subject), Squiggle is completely domain-independent and can thus serve as a *framework* to build domain-specific

search applications. In essence, Squiggle is not a search engine itself, but it allows users to customize their own engine on the basis of a particular domain knowledge, as will be explained in section 5.

Squiggle is designed to provide both syntactic and semantic indexing and searching primitives, seamlessly combining the speed of syntactic search tools with improved recall and precision, based on the ability to assign alternative designations and wordings in multiple languages to their meaning. Among the constituents of Squiggle, Sesame[7] [12] is used to store and query semantic information constituting the knowledge base, described in RDF/OWL with regard to the SKOS model, whereas the syntactic search engine Lucene[8] [13] is used, among other things, to quickly perform text searches in literals, which is something that semantic search tools typically cannot do well. Therefore the described architecture lends itself well both to overcome the limitations of purely syntactic approaches and to improve the performance of semantic engines.

The main components of Squiggle are sketched in figure 1. In the following, we provide a brief description of Squiggle's Conceptual Indexing (§4.1) and Semantic Search (§4.2) capabilities. Moreover, in §4.3, we illustrate how to add plug-in extensions to customize Squiggle and strengthen its potentialities.

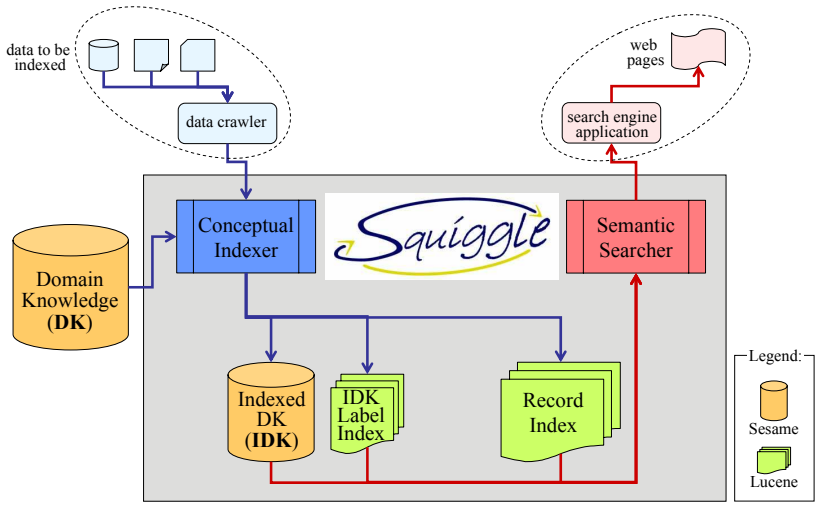

Fig. 1. The architecture of Squiggle framework

4.1 Conceptual Indexing

Indexing can be defined as a complex process that transforms the input information into a format that allows an efficient and easy search. *Conceptual Indexing* is an indexing process that, in addition, tries to semantically annotate contents with the corresponding "concepts".

[7] http://openrdf.org/
[8] http://lucene.apache.org/

Squiggle's semantic annotation process is very minimalist; it expects resources to be already annotated with keywords and it searches in the Domain Knowledge repository for concepts whose SKOS labels match those keywords. If a single concept is retrieved, then the resource is annotated with it. Otherwise, we use heuristic approaches to identify the closest concept(s). If also these heuristics don't work, some manual annotation is required. In any case, the identified concepts are copied from the Domain Knowledge and inserted into the Indexed Domain Knowledge for efficiency reasons[9].

The default implementation of Conceptual Indexing is straightforward: first the input information is scanned and analyzed in order to identify and extract the concepts that characterize it (the *semantic annotation process* described above); then, these concepts are stored in the index for subsequent search and retrieval (*indexing process*). The index consists of three parts:

1. the main part is a Lucene index (cf. Record Index in fig. 1), that indexes all the textual parts and the URIs of the identified concepts, which are helpful in the syntactic searching phase;
2. the "semantic" part is a Sesame repository (cf. Indexed Domain Knowledge or IDK in fig. 1), that contains all the triples that the search engine will need at runtime, when answering final user's queries;
3. another smaller Lucene index (cf. IDK Label Index in fig. 1) with the role of supporting Sesame repository, that indexes all the labels contained in IDK, in order to speed up their retrieval at search-time[10].

4.2 Semantic Search

Squiggle can be used to perform searches both at the syntactic and at the semantic level. The employment of Lucene in a "traditional" way (i.e., to index text) assures the classical syntactic search. Moreover, the previous Conceptual Indexing phase enables also semantic searches.

Every time a user submits a query, Squiggle's Semantic Searcher analyses it and tries to identify the ontological elements – contained in Indexed Domain Knowledge repository – that can be related to the request. Then, it is able to "suggest" to the user the potential *meanings* of his query that it recognized; the user is therefore presented with both the results of the syntactic search and the available meanings extracted from the query, which can help him to refine his request, "disambiguating" among its the possible acceptations.

The MeaningSuggester is the abstraction we use to identify the "tool" able to find ontological elements that have some semantic relationship with the search. A MeaningSuggester accesses the Indexed Domain Knowledge repository in order to extract concepts or instances thereof (from now on called *meanings*) that answer the user query. Once a specific meaning is identified (i.e., the

[9] IDK is therefore the minimum useful subset of Domain Knowledge repository; the creation of this repository aims at speeding-up searching time, in case DK contains millions of triples, the majority of which is not used at run-time.

[10] Being very fast in retrieving data in response to any syntactic search, Lucene can help in indexing all the literals contained in the "semantic" domain knowledge.

user disambiguates by selecting one of the suggestions), a semantic search consists in looking up the indexed resources whose semantic annotation corresponds to that meaning.

But there is much more: when a user query is reconducted to a specific meaning, Squiggle's Semantic Searcher is able not only to look up resources semantically related to that meaning, but also to seek other concepts that could be of interest for the user. This is possible because Squiggle's Semantic Search can navigate across the graph of interconnected elements of the domain ontology, following "semantic paths" denoted by relations and attributes.

A MeaningSuggester is therefore a common pattern that can have multiple behaviors (i.e., different execution semantics), since it can find meanings according to different criteria. For example, the MeaningSuggester may return the meanings that match a given query after expansion of the query terms to their aliases (playing on the existence of a `skos:prefLabel` and multiple `skos:altLabel`s for each concept); furthermore, the MeaningSuggester may return meanings that are related to the query terms by conceptual narrowing (`skos:narrower`) or broadening (`skos:broader`) in the domain knowledge and so on.

Squiggle framework is designed with a bunch of ready-to-use MeaningSuggester behaviors, since it is based on SKOS model and therefore can exploit SKOS properties (besides the previously cited ones, the MeaningSuggester *knows* also relations like `skos:subject`, `skos:related`, `skos:relatedPartOf`, etc.). In table 1, we give more details on the process Squiggle uses to "suggest" meanings.

4.3 Building extensions for Squiggle

As described above, Squiggle is designed by following a model-driven approach that uses SKOS as model. Therefore both the Conceptual Indexing and the Semantic Search do not rely on concepts or properties defined in the domain ontology, but on SKOS.

However, to build a more powerful semantic search engine, knowledge about the domain can be inserted into the system, in order to exploit Squiggle's capabilities in indexing and searching. The flexibility of the Squiggle framework is based on its ability in adapting to any real case: whoever needs to build a search engine in a particular domain can easily employ Squiggle and extend it by creating ad hoc *plug-ins*, gaining not only in recall and precision during searches, but also in a more reliable indexing. Therefore, semantic annotation and meaning suggestion plug-ins should be added as part of the deployment.

As we reported in section 4.1, Squiggle has a minimalist semantic annotation plug-in that is sufficient to index resources for which semi-structured annotation are already available (i.e., ID3 metadata in mp3 files). For all other cases, we provide a way for integrating custom semantic annotation techniques as plug-ins. For instance, acoustic fingerprints (e.g. MusicIP's PUIDs[11]) identify audio files, basing on the contained audio data with good tolerance to encoding, resampling and noise. Combining such smart machine technique with smart data

[11] `http://wiki.musicbrainz.org/HowPUIDsWork`

Semantic Relationship	**Representation**	**Execution Semantics of** MeaningSuggester
homonymy	same SKOS label for different OWL classes	given a term, it returns the SKOS `prefLabel` of each OWL class that has the term as SKOS label
pseudonymy	SKOS `altLabel`s of the same OWL class	given a SKOS `altLabel` of an OWL class, it returns all its SKOS labels
synonymy	different SKOS labels of the same SKOS concept	given a SKOS concept, it returns all the SKOS labels of the same SKOS concept
	SKOS labels of OWL classes related by equivalence property	given an OWL class, it return all the SKOS labels of the equivalent OWL classes
hyperonymy	SKOS concepts related by SKOS `broader` relationship	given a SKOS concept, it returns the SKOS concepts related via SKOS `broader` property
	entailment among OWL classes	given an OWL class, it returns its super-classes
hyponymy	SKOS concepts related by SKOS `narrower` relationship	given a SKOS concept, it returns the SKOS concepts related via SKOS `narrower` property
	entailment among OWL classes	given an OWL class, it returns its sub-classes
meronymy	SKOS concepts related by SKOS `relatedPartOf` property	given a SKOS concept, it returns the SKOS concepts related via SKOS `relatedPartOf` property
generic semantic relationship	SKOS concepts related by SKOS `related` property	given a SKOS concept, it returns the SKOS concepts related via SKOS `related` property

Table 1. The default MeaningSuggester execution semantics in Squiggle.

(e.g. MusicBrainz), deriving author and title is almost automatic (cf. Squiggle Music deployment in §5.2).

As we reported in section 4.2, the predefined MeaningSuggester, provided by the Squiggle framework, reflects the most common needs, since it exploits general semantic relations. On the other hand, when building a search application within a specific domain, managing other relations is useful too. This aim is easily achieved by defining one or more domain-aware MeaningSuggester behaviours: each one of them will follow a specific semantic "path" connecting resources. For example, in Squiggle Music application (see below §5.2), we created a SongArtistMeaningSuggester that finds the songs connected through the domain property `music:performedBy` to a given artist.

5 Squiggle real-world deployments

In order to prove the feasibility of our approach, we briefly present some test beds. We successfully developed some search engines on top of the Squiggle framework, in different application fields. This section illustrates how Squiggle can be instantiated to a specific domain-aware semantic search engine.

To build a domain-specific search engine, the first indispensable ingredient is, of course, the domain knowledge, which is essential to the indexing process, because it is the basis on which concepts can be identified within the input information. We assume that this knowledge is systematized and formalized in a machine processable structure; in particular, in RDF/OWL with regards to SKOS model. In our architecture, the formalized knowledge is stored in a Sesame repository called Domain Knowledge (see figure 1). This repository is not a proper part of Squiggle, but it is evident that no application can be built without it.

Once the domain knowledge is formalized, Squiggle can be further configured with domain-specific extensions, as explained in §4.3.

For what regards searching time, the choice we made was to put user experience at the center of our investigation: we designed the search interface to be the simplest possible and the query system to be the most intuitive for the user. Therefore, we chose to have a very minimalistic query form that accepts keyword-based requests, like the most common search engines we are used to, hiding behind this simple interface the elaborations to analyze the query and to retrieve results of interest. Furthermore, the additional supporting "suggestions" the user receives in response to his queries are displayed in clear and non-invasive side-boxes.

The aim of this simplification of the user interface is to provide fresh and innovative features without overloading their presentation and maintaining an immediate and clean visual experience.

5.1 Squiggle Ski

CEFRIEL, as Official Supplier of Applied Academic Research for Torino 2006 Olympic Winter Games, caught the opportunity to demonstrate Squiggle in the context of CEFRIEL's activities related to Torino 2006[12]. Our aim in deploying Squiggle Ski, a service available on CEFRIEL's portal, was to help the international public of Torino 2006 in finding images of the athletes involved in the alpine skiing races.

Precision and recall of popular search engines, which are very good at retrieving Web pages, *drop down when dealing with images*. A simple but effective example is searching for an athlete and a discipline, without using the athlete's mother-tongue. For instance, searching for "`Hermann Maier`" (the Austrian downhill champion) and "`discesa libera`" (the Italian expression for "downhill" discipline) an average of ten images is retrieved and half of them are not relevant. On the contrary, searching for "`Hermann Maier abfahrt`" (the German word for "downhill") around one hundred images are retrieved and almost all of them are relevant. If users had the knowledge of the names of a discipline in many languages, they could write a query like "`Hermann Maier ("discesa libera" OR downhill OR abfahrt OR descente)`" obtaining hundreds of results, the first two hundreds of which are relevant.

[12] See also `http://www.cefriel.it/press/olimpiadi2006.html?lang=en`

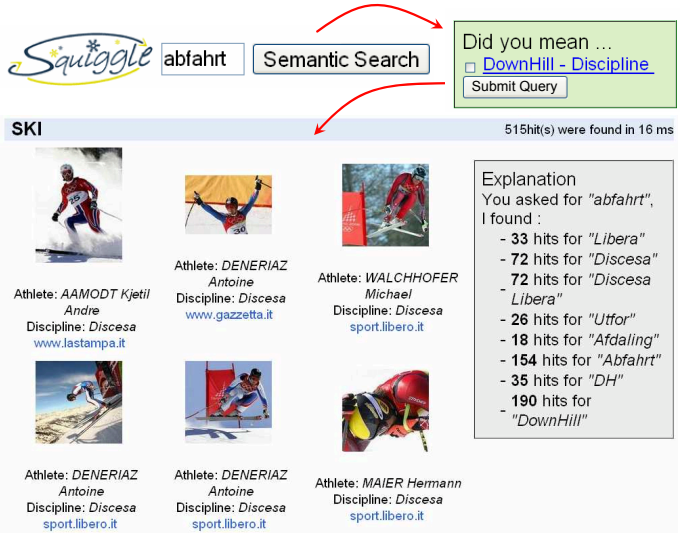

Fig. 2. A screenshot of Squiggle Ski

These examples simply highlight that *implicit language assumptions*, which hold in searching for Web pages, *must be reconsidered when searching for images*. In other words, searching for Web pages using a given language has, in general, a strict implication on the language of the page, whereas searching for images has not. *A semantic search engine can handle this implicit knowledge* by means of an ontology that represents the domain, both conceptually (including OWL classes for athletes, resorts, disciplines and the respective instances) and lexically (including multiple SKOS labels in multiple languages for each concept).

In order to instantiate Squiggle Ski, we built a domain ontology partially by hand, developing a small multilingual taxonomy[13] of the disciplines in the sectors of alpine skiing, and partially by collecting information on the FIS-Ski web site[14], from which we collected all the names of the athletes that got a podium, as well as all the events in the last three years and the relationships with the nations that hosted them, the top three athletes of the event and the type of discipline.

Then we built an experimental focused crawler that exploits the knowledge in the ski ontology to collect images of skiers from sport news Web sites all over the world. The awareness about all relevant terms (names of athletes, disciplines, places, etc. with possible alternative labels in different languages) helps both the focused crawler to filter the appropriate photos and the Conceptual Indexer to

[13] Disciplines' concepts have labels in English, Italian, German, French, Swedish, Norwegian and Finnish.

[14] `http://www.fis-ski.com/`

semantically annotate them before the indexing process. The crawler was tested for a couple of month before Torino 2006 and it was on for the entire Torino 2006 event. It collected more than 1800 images.

Squiggle Ski is on-line at `http://squiggle.cefriel.it/ski`; during Torino 2006 event, it was visited by almost one thousand visitors searching for the various athletes that won a medal in the alpine-skiing races. When you open the home page, you are presented with an ordinary search box. If you try searching for "`Herminator abfahrt`" (being "Herminator" a nickname for Hermann Maier and "abfahrt" the German for downhill), you receive a plain syntactic search, and in a box on the right Squiggle Ski asks if you mean the athlete "Hermann Maier" and the discipline "downhill". If you eventually follow Squiggle Ski suggestions, all the images of Hermann Maier in a downhill race contained in the repository are retrieved, disregarding the language used in the initial query; an explanation box shows how Squiggle Ski expanded the query to achieve the result (see figure 2 for an example).

5.2 Squiggle Music

Squiggle Music is an instantiation of Squiggle framework in the music field. We noticed that both very diffuse media-players and popular sites for buying music fail to retrieve tracks when alternative wordings or translations are used (e.g. searching "rhcp" does not always retrieve the list of all Red Hot Chili Peppers tracks in the repository).

Squiggle Music indexes audio files (mainly mp3 files) enriching them with information about authors, song titles and music genres. Squiggle Music is publicly available at `http://squiggle.cefriel.it/music`.

The sources of these data are two freely available meta-databases developed and maintained by web communities: MusicMoz[15] and MusicBrainz[16]. The latter contains very comprehensive information concerning names of music bands and titles of tracks as well as associations between different bands/artists. The former is a smaller set of XML documents that, however, also includes a taxonomy of musical styles and associations between bands/artists and styles. To build the Domain Knowledge repository we merged the data contained in these repositories, creating a SKOS-based RDF/OWL ontology; the resulting knowledge base contains more than two million triples.

In order to employ this huge knowledge while indexing, we made use of an interesting case of audio recognition technique; MusicBrainz, in fact, for each song, offers its TRM id[17]. TRM is an audio fingerprinting technology that generates a unique fingerprint for an audio file based on an analysis of the acoustic properties of the audio itself. This "barcode" is independent from the particular file format and can be extracted from almost any audio file; each unique fingerprint can therefore be compared with existing databases of fingerprints in order

[15] `http://www.musicmoz.org/`
[16] `http://www.musicbrainz.org/`
[17] `http://www.relatable.com/tech/trm.html`

to identify the specific musical track precisely. Therefore, using tools like QuickNamer[18], a small stand-alone application that is able to calculate the TRM of a file, and searching in MusicBrainz database for matching, it's possible to put an mp3 file in relation with the song's metadata[19].

Combining the *smart data* from MusicBrainz and MusicMoz with a *smart machine* like QuickNamer, we built an automatic semantic annotator that acts as a domain-dependent plug-in of `Squiggle` framework during the *Conceptual Indexing* phase. This annotator is therefore able to add to each file all its metadata (artist, song title, etc.).

From the final user's point of view, the search facilities offer many interesting options. Besides the usual "suggestion" of meanings through the analysis of user's query, Squiggle Music is able to perform a query expansion and to present the user with other results that could be of his interest. This is possible because the *Semantic Search* of Squiggle Music is strongly founded on the different (default and ad hoc) MeaningSuggester behaviours: Squiggle Music can suggest to the user related artists when searching for a performer, songs by the same artist when looking for a song, broader and narrower styles when asking for a music genre.

For example, if the user query is "`rhcp`", the system suggests "`Red Hot Chili Peppers`" as one of the possible meanings ("rhcp" being a known acronym); selecting the proposed meaning results in better recall, since only few songs are likely to match the search string syntactically. A query for the "`Queen`" band will propose, e.g., related artists such as Freddie Mercury (who was part of the band). Finally, asking for "`Celtic`" genre, Squiggle Music can propose to widen the query suggesting "`World`" music, a broader meaning, and "`Celtic Pop`" style, a narrower meaning.

6 Future works

The next step in `Squiggle` evolution is to make it accessible over an API, in order to let other (even already existing) applications to integrate `Squiggle` semantic search functionalities in their architecture. E.g., the `Squiggle` API will let applications submit concept-based queries and will return a set of matching multimedia results.

Moreover, we plan to integrate `Squiggle` with `SOIP-F` [20][14], our solution for building semantic based portals. This integration aims at making `Squiggle` a retrieval component for a semantically enabled multimedia presentation system, so that media can be selected, adapted and organized in a coherent and homogeneous presentation that improves user experience in searching and browsing.

[18] `http://phonascus.sourceforge.net/`

[19] An alternative of TRM ids is today represented by PUIDs (see `http://www.musicdns.org/`); MusicBrainz is currently enclosing also this kind of acoustic fingerprint in its repository.

[20] `http://seip.cefriel.it`

Finally, in order to better highlight the value-added brought by Squiggle, we plan to use a formal evaluation framework to assess our solution. We aim at setting up a scheme to take the necessary measurements of precision/recall and to compare Squiggle behavior against other search engines. The main obstacle we meet in designing this evaluation framework is the difficulty to compare Squiggle with other search engines, because of the lacking of "homogeneous" systems, since Squiggle – or rather each instance of Squiggle framework – is strictly domain-dependent and cannot be easily compared to general-purpose search engines.

In additions, we are planning some user trials to measure the usability of our solution; our (subjective) opinion is that Squiggle is a user-friendly tool: the user interacts with Squiggle exactly as he/she does with any other search engine, but he/she can have a better experience because of the suggestions of meanings.

7 Conclusions

In this paper, we presented Squiggle, a Semantic Web framework that eases the deployment of semantic search engines in specific applications. We enlightened how the employment of Semantic Web technologies to the development of search engines provides real benefits to end users, enabling an easier and more effective access to information; a semantic search engine, in facts, improves current syntactic engines in terms of both precision and recall, thanks to an explicit characterization of the domain at lexical and conceptual level. Semantic Web technologies show their whole potentialities in the expansion of queries to include related meanings: a semantic search engine built on Squiggle appears to be more usable, in that users are supported with semantic "suggestions", as our test-beds demonstrate at a glance.

Moreover, we designed Squiggle keeping in mind the particular needs of searching when dealing with multimedia contents. In particular, we forecast a larger adoption of smart machines to process media, in order to promote a better generation and aggregation of smart data while exploiting media-dependent capabilities. The extensible nature of Squiggle fully enables the joint employment of smart machines and domain ontologies. It is worth noting that one of the major cons of a semantic approach is the availability of domain ontologies that formalize a specific knowledge in a structured way. With this regard, we strongly wish for a large-scale adoption of SKOS model for the development of ontologies, because we believe that SKOS represents a good trade-off between the expressiveness of an ontological language and the simplicity of organization systems (like categorizations and taxonomies).

Finally, we admit that a semantic search engine developed with Squiggle is strongly domain-dependent and cannot compete with general-purpose search engines; however, we definitely believe that such an approach provides better results, especially when dealing with multimedia contents, because a focused tool better meets specialized needs, helping you in finding what you're really looking for.

Acknowledgments

The research has been partially supported by the NeP4B project, funded by Italian Ministry of University and Research (MIUR project, FIRB-2005). We would like to thank our former colleagues Davide Martinenghi and Francesco Dolcini for their help and support in designing and developing Squiggle.

References

1. Sergey Brin, Lawrence Page: The Anatomy of a Large-Scale Hypertextual Web Search Engine. Computer Networks and ISDN Systems **30**(1–7) (1998) 107–117
2. Tim Berners-Lee: Semantic Web Road map. Available on the web at `http://www.w3.org/DesignIssues/Semantic.html` (1998)
3. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project (1998)
4. Miles, A., Brickley, D.: SKOS Core Guide, W3C Working Draft. `http://www.w3.org/TR/swbp-skos-core-guide` (2 November 2005)
5. Patel-Schneider, P., Hayes, P., Horrocks, I.: OWL Web Ontology Language, W3C Recommendation. `http://www.w3.org/TR/owl-semantics/` (10 February 2004)
6. Jos Kahan, Marja-Riitta Koivunen, E.P., Swick, R.R.: Annotea: An Open RDF Infrastructure for Shared Web Annotations. In: Proceedings of the WWW10 International Conference, Hong Kong. (2001)
7. Kalyanpur, A., Parsia, B., Hendler, J., Golbeck, J.: SMORE - Semantic Markup, Ontology, and RDF Editor. (2002)
8. Atanas Kiryakov, Borislav Popov, Ivan Terziev, Dimitar Manov, Damyan Ognyanoff: Semantic Annotation, Indexing, and Retrieval. Elsevier's Journal of Web Semantics **2**(1) (2005)
9. R.V.Guha, Rob McCool: TAP: A Semantic Web Platform. in proceedings of the Eleventh International World Wide Web Conference (WWW2002), May 7-11, Honolulu, Hawaii, USA (2002)
10. Wray Buntine: Open Source Search: A Data Mining Platform. Invited talk given at the Fourth IEEE International Conference on Data Mining, Brighton, UK. (2004)
11. Thomas Kamps: ConWeaver. See `http://www.conweaver.de` (2005)
12. Arjohn Kampman, Frank van Harmelen, Jeen Broekstra: Sesame: A generic architecture for storing and querying rdf and rdf schema. in proceedings of ISWC 2002, October 7 - 10, Sardinia, Italy (2002)
13. Otis Gospodnetic, Erik Hatcher: Lucene in action. Manning Publications (2004)
14. I. Celino, E. Della Valle: Multiple vehicles for a semantic navigation across hyper-environments. In proceedings of the Second European Semantic Web Conference, ESWC2005 (2005)