The Beagle^{++} Toolbox: Towards an Extendable Desktop Search Architecture

Ingo Brunkhorst, Paul - Alexandru Chirita, Stefania Costache,
Julien Gaugaz, Ekaterini Ioannou, Tereza Iofciu,
Enrico Minack, Wolfgang Nejdl and Raluca Paiu

L3S Research Center / University of Hanover
Deutscher Pavillon, Expo Plaza 1
30539 Hanover, Germany
{brunkhorst, chirita, costache, ...}@l3s.de

Abstract. The rapidly increasing quantity and diversity of data stored on our PCs made locating information in this environment very difficult. Consequently, recent research has focussed on building semantically enhanced systems for either organizing or searching data on the desktop. Building on previous work, in this paper we present the Beagle^{++} toolbox, a set of extendable building blocks for implementing such a system. The corresponding modular desktop search architecture integrates our previously developed metadata generators and ranking components, uses an RDF database to share data between components, and can easily integrate other external components to improve desktop search quality. Additionally, we provide implementation details about all our current components, how they interact with each other, and how to install the complete system on top of a Linux distribution.

1 Introduction

Most leading search companies have recently created and offered free desktop search applications. For managing your desktop's several gigabytes of data, these tools build and maintain an index, typically by collecting two types of information: (1) file and directory names and (2) content of supported documents. At present, only very few of the deployed desktop search engines also collect very basic metadata information, such as titles, authors or comments, usually already contained in the files being indexed. Yet very few people spend time annotating their documents, and thus this functionality provides only a limited improvement over text-based search.

On the other hand, the desktop environment provides a lot of semantic relationships which should be used to enhance search on the desktop. When searching their desktops, users usually search for very specific items they have already worked with (otherwise why not search the Web?), such as an email from a colleague, a photo or movie taken on a special occasion, the contact information of a friend, etc. Satisfying such a search requires semantic information.

Our previous work within the Beagle^{++} context [4, 6] has started to investigate means to represent and use this semantic information, and has introduced several innovative *modules* to enhance a desktop search application. In this paper we focus more on the architectural aspect of such a toolbox, and discuss how to utilize these modules into an extendable full-fledged semantic desktop search application. We describe the different types of modules, ranging from metadata generation to indexing and ranking desktop resources, as well as the communication channels and interfaces they use. We also discuss the issues arising from the need to merge metadata from different sources, and discuss the different parts of a desktop ontology as well as functionality for entity identification in our system.

Specifically, the next section presents the modular toolbox architecture we rely upon, focusing on the set of metadata extraction and generation modules. Section 3 then discusses the desktop ontology used by all modules to ensure coherent metadata, and proposes solutions for the problem of merging the metadata produced by the different modules. Section 4 focuses on related work, section 5 summarizes the main ideas of this paper and presents future work and research issues.

While this paper focuses on the Beagle^{++} toolbox, part of this work is being done in the context of the NEPOMUK project[1], which aims at creating the *Social Semantic Desktop*. It does so by extending a regular *Desktop* in two directions: (1) towards a *Semantic* environment by enabling the use of semantic descriptions of desktop objects and their relationships, and (2) towards a *Social* environment by enabling the communication with other desktops connected in a social network. Our Beagle^{++} toolbox is a first step towards this vision, and will be integrated with contributions from our partners into the future *Social Semantic Desktop* NEPOMUK project infrastructure.

2 The Beagle^{++} Modular Desktop Search Architecture

As basis for our Beagle^{++} environment we use the open source Beagle desktop search engine[2] for Linux, which we extend with advanced searching and ranking capabilities exploiting semantic information. Figure 1 illustrates the overall Beagle^{++} architecture. White boxes represent the original Beagle components, gray colored ones represent our extension modules and white/gray boxes correspond to Beagle components which we modified. In the following, we describe this architecture in detail, and then present its components for metadata generation, storage, indexing and ranking.

2.1 Architectural Overview

The *Beagle Control Process* is the core component of the Beagle architecture and is in charge of dispatching indexing and searching requests to the appropriate

[1] `http://nepomuk.semanticdesktop.org/`
[2] `http://www.beaglewiki.org/`

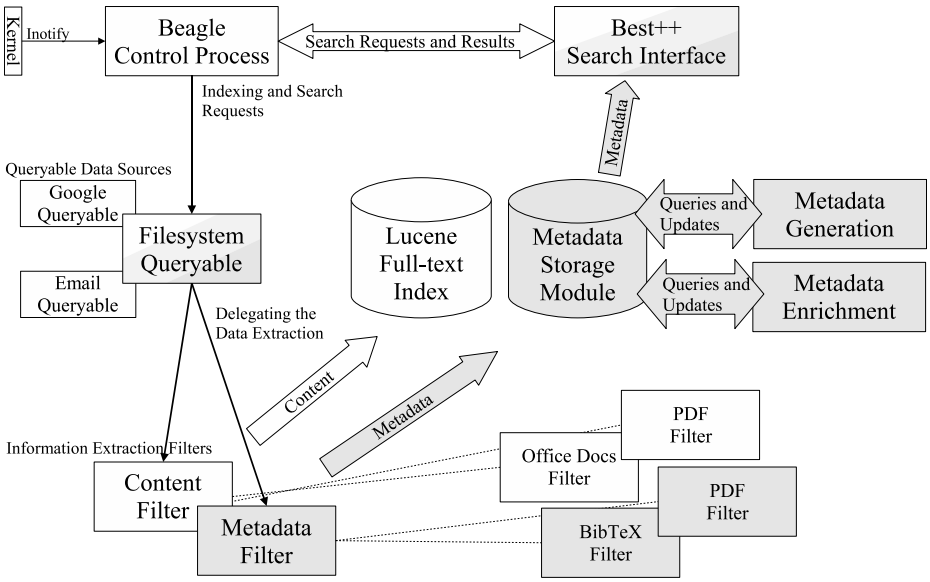

Fig. 1. Beagle^{++} Architecture Overview

modules. All file system events (files being created, modified or deleted) caught by the Inotify-enabled Linux Kernel are sent to the Beagle Control Process, which generates the requests to update the information in the index. For each request, the *Filesystem Queryable* selects an appropriate *Filter* for extracting the content, as well as the metadata of the files. This extracted information is indexed by the *Lucene Full-text Index* module and stored in the *Metadata Storage Module*. Each Filter processes a specific type of file, identified by filename extension or MIME type (e.g. application/pdf). Based on the type of information extracted, the Beagle^{++} Filters are classified into two categories: *Content Filters* and *Metadata Filters*, the latter representing one of our extensions to the original Beagle architecture.

Best^{++} implements the search interface of our desktop search engine and is responsible for the routing of search requests from the user to the Beagle Control Process, which hands them to the Queryable Data Sources. These try to find relevant results matching the queries both in the Full-text Index and in the Metadata Storage Module. The search results gathered from the Full-text Index are merged by the Control Process into one list of result documents, which is enriched by Best^{++} with metadata and ranking information extracted from the Metadata Storage Module.

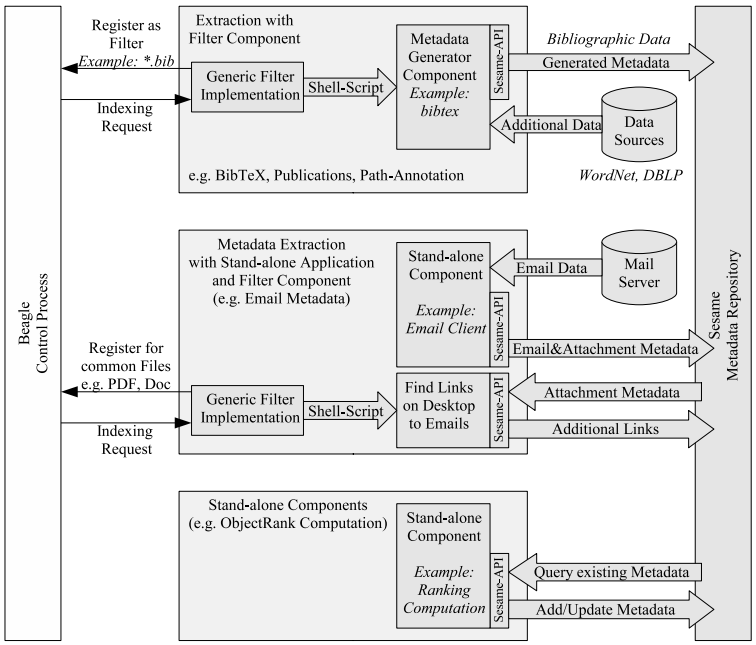

Fig. 2. Beagle^{++} Metadata Components

2.2 Metadata Extraction and Generation Components

The original Beagle architecture uses *Filters* to extract content from specific types of documents. The actual creation of a text-based representation of the content needed for indexing is usually not done directly in Beagle, but delegated to third-party tools. When new documents are to be indexed, Beagle searches its library for suitable Filters for extracting the content, as well as basic properties, for example title and author in the case of PDF documents. For metadata extraction in Beagle^{++} we use a similar approach: We extended the Filter interface with a third functionality, which in addition to the content extraction also extracts the metadata from the corresponding documents. Figure 2 presents the different strategies for metadata creation which are currently implemented in Beagle^{++}. Most of the implemented metadata processing facilities for Beagle^{++} can be classified into three different approaches: First, extraction via a single Filter component, integrated into the original Beagle's filter architecture; second, metadata creation with a stand-alone tool, which is supported by a Beagle-integrated component with additional information; third, a pure stand-alone metadata component working solely with data from the Metadata Storage Module, and run at regular intervals. The last strategy is used for RDF indexing and ObjectRank computations as described in Sections 2.4 and 2.5.

The Beagle^{++} metadata creation components are composed of the *Generic Filter Implementation*, the actual *Metadata Generator Component*, and an op-

tional *Stand-Alone Component*. The Generic Filter is a wrapper that registers itself as a Filter in the Beagle infrastructure and receives events related to files of the supported types. Instead of processing metadata on its own, it delegates the actual work to beagle-independent tools, which are usually invoked using a shell script providing the URI of the document to be processed. Beagle^{++} Components are not limited to using the data extracted from documents, but often rely on additional data sources as well. Stand-alone tools gather metadata on their own, which are then used by other components. The following components are currently implemented:

Metadata Extraction using a single Filter component:

Path Annotation. Folder hierarchies are barely utilized by the search algorithms, in spite of the often sophisticated classification hierarchies users construct. For example, pictures taken in Hanover could be stored in a directory entitled "Germany", so it would be useful if we could use this information for search. The path annotation component annotates files with each token in their file path, as well as additional semantic information provided by the WordNet system[3], such as synonyms, hyponyms, hypernyms, meronyms and holonyms [4].

Scientific Publications. In the research community, many papers are available in PDF format. Although PDF allows basic metadata annotations like title and authors, this is rarely used. We developed a tool which extracts metadata from files using the publicly available Citeseer[4] and DBLP[5] databases. Its main idea is to first pre-compute word vector representations of the titles stored in these databases, such that we can search for title candidates consisting of the first words of a publication. If the title is found, we query for additional information like authors or conference, year of publication, etc. from these databases.

Publication Bibliography Data. BibTeX documents are common for storing publication references. It is also common to share these files with co-authors or the local group of researchers. Our tool creates RDF metadata from each BibTeX file found on the desktop, including keys, titles, authors, and other annotations available in the BibTeX format. The data created by this module complements the metadata extracted by the Scientific Publications module.

WebCache. This component facilitates the users' search for web pages by starting from a familiar or prominent web site. We broadened the notion of "visited link" by defining it as a web page that was previously visited by the user (the link's target page is present in the browser cache). At run-time, for enhancing navigation, all these visited links are highlighted. These metadata are created for every web page in the cache, containing the links that have been visited from that page, as well as the in-going links from which the user could have arrived to it (inverse of a visited link).

[3] `http://wordnet.princeton.edu/`
[4] `http://citeseer.ist.psu.edu/`
[5] `http://dblp.uni-trier.de/`

Metadata Extraction using a Stand-Alone Application, with and without support from a Filter component:

Emails and Attachments. We developed two components following this approach: (1) The email client and (2) the beagle-integrated Filter for the "email attachment - stored file" link creation. The email client is a simple stand-alone process that monitors the users' email account on any POP3 or IMAP-capable mail-server. For each email object an RDF description is created with information from the header, body and the available attachments. The second component uses a Filter integrated into Beagle to check for each recently stored document if it was contained in the attachment of a received email. Connections between a document and an email attachment are described according to our ontology and made persistent as a link between resources in the Metadata Storage Module.

2.3 Metadata Storage

While our approach for metadata extraction and generation allows us to easily incorporate additional generators and filters to cover more types of desktop objects, managing all these metadata becomes a major issue. We therefore designed a Metadata Storage Module to handle all these metadata in a uniform manner, keeping in mind that the metadata will come from different applications and different sources. Also, the Metadata Storage Module has to be able to incorporate metadata from generators and filters, designed and developed by other partners in the NEPOMUK project.

The repository is the core component of this module, which is realized as a Sesame repository [3] running as a web application under Apache Tomcat, backed by a MySQL database for the actual data storage. Metadata can be accessed in two ways, either through the web-services provided by the Sesame application under Tomcat, or through the *Sesame-API for Beagle^{++}*. This is a Java-API which provides a subset of the functionality of the Sesame API with specific Beagle^{++} customizations. The Beagle^{++} Metadata Storage Module does not only unify metadata from different sources and applications, but also provides a generic mediator framework for manipulating these metadata. Currently, Beagle^{++} uses the Metadata Storage Module to perform two essential processes. The first process is the ObjectRank computation, explained in Section 2.5, and the second one is the Entity Identification algorithm, explained in Section 3.2.

The Metadata Storage Module provides a solid ground for incorporating advanced functionalities into Beagle^{++}. The most important functionality is that it acts as a mediator between the modules which create metadata and the modules using it, ensuring that Beagle^{++} remains a modular toolbox. For example, consider implementing a new filter for extracting information from a specific type of desktop objects. Incorporating this filter into Beagle^{++} requires only the use of the appropriate methods from the Beagle^{++} Sesame-API which insert its metadata into the Sesame repository. Another advanced functionality lies in the ability to easily design and execute algorithms on the collected metadata. Such

algorithms directly query the metadata found in the Sesame repository, process them and store any possible results back to the Sesame repository.

2.4 Indexing

Basic Lucene Indexing.

Beagle uses Lucene[6] as a back-end high performance full-text search engine. Lucene is a multi-purpose information retrieval library for adding indexing and search capabilities to various applications. Its indexing process breaks down into three main operations: converting data to text, analyzing, and storing it into its index structures. As most full-text search engines, Lucene implements an extended version of the vector space model, which supports fragmentation of the vector space into separate namespaces (denoted as "fields" in the actual implementation). Thus, a term does not only represent indexed words of the document's body, but also states the context in which the word appears. The "default" context is the full-text of the document, whereas other contexts often describe metadata such as author or title. Two terms sharing the same literal are treated as non-equal if they appear in different fields. Finally, Lucene provides an efficient way to search for phrases which is not directly supported by the vector space model. Lucene maintains a "positional index" which contains for each term the exact positions in each document. Positional indices add another form of "context" besides the notion of separate term fields, namely the proximity of terms in one single field.

Beagle (i.e. its Lucene component) creates two parallel indices for each document. The first index contains full-text fields for storing keywords and several fields for metadata, one per property. The second index contains property fields for metadata. The two indexes are used for allowing querying different fields at the same time. Beagle does not search for context information outside the documents, so usually almost no metadata is present in a standard Beagle installation.

RDF Indexing. Metadata indexing can be accomplished in several ways. The approach used in Beagle defines one field per metadata predicate (e.g., one for "description", one for "title", etc.). While the idea to use separate fields is well suited for directly annotated metadata, it is not suitable for *metadata paths*. Furthermore, the ability to rank documents by metadata literals is lost because the TFxIDF measure used by the vector space model does not span across fields.

In Beagle^{++} we therefore use one single field for all metadata associated to a document. For each directly associated statement, we store both predicate and object of the statement as text in this field. This makes storage independent of any underlying RDF schema, as the predicates are represented as terms rather than field names. To describe more complex contexts, we store *predicate paths*, representing the properties which are not directly annotated to the document, but can be reached by following a path in the RDF graph starting from the document node via subject–predicate–object connections. RDF fragments can

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

then simply be matched using phrase queries on the metadata field, and results are returned almost instantaneously, as querying reduces to lookup operations in the Lucene index.

Path Materialization and Updating. As described above, our extended documents consist of full-text plus associated RDF metadata. This associated RDF metadata graph has to be materialized in a full-text search infrastructure in order to guarantee efficient retrieval and integrated search. For example, given the statements $[R1\, p1\, R2]$ and $[R2\, p2\, R3]$, we are able to address $R3$ from $R1$ via the path $p1/p2$. In a relational database, path traversal can be implemented by performing join operations on the RDF resources, which are costly operations especially when performed very frequently within a large scale data set. We avoid these joins in our full-text search environment by materializing paths, creating "expanded" statements like $[R1\, p1/p2\, R3]$ and associating them to $R1$, as any other directly annotated metadata.

Path materialization is only feasible for a limited number of paths. We are currently experimenting with different options, restricting path materialization to start from full-text documents and stop as soon as we exceed a certain path depth or reach another full-text document. This is because we want to re-index only documents for which metadata has changed, and we can find out which are these documents just by searching the Lucene index.

2.5 Ranking Module

Overview. The *Ranking Module* provides enhanced ranking capabilities for our desktop search engine. Unlike current desktop search engines, which rely only on basic TFxIDF measures for ranking the search results, we use a combination of the TFxIDF scores provided by the original Beagle application with additional ObjectRank values. As stated in [6], ranking computation on the desktop performs poorly without contextual information, which is used to reconstruct the links among the resources. Therefore, our ObjectRank computation relies on the input provided by the Metadata Storage Module and the Metadata Generator Modules.

The ranking module consists of two components, (a) one being in charge of the ranking computation (*ObjectRank Computation Component)* and (b) one handling the combination of the TFxIDF and ObjectRank scores (*Re-ranking Component*). The ObjectRank computation is transparent to the applications using the ObjectRank values: the computed scores become available as soon as the generators and extractors have written their metadata into the Metadata Storage Module.

ObjectRank Computation Component. The ObjectRank computation is based on the Beagle^{++} ontologies and on the corresponding authority transfer schema. The ontologies are extended by adding weights and edges in order to express how importance is propagated among the entities and resources described by the ontology. Weights and edges represent authority transfer annotations extending the ontology with the information we need to compute ranks for all instances of the classes defined in the ontology.

As illustrated in [2], the authority transfer schema graph and the schema graph (here our Beagle^{++} ontologies) reveal some significant redundancy. In order to reduce that to a minimum we define the authority transfer schema graph by annotating our Beagle^{++} ontologies with forward and backward weights to edges that transfer authority. An RDF graph that conforms to this schema is the main configuration for the ObjectRank Computation Component. For the ranking computation, the first component uses the PageRank formula

$$r = d \cdot A \cdot r + (1 - d) \cdot e \tag{1}$$

applying the random surfer model and including all nodes in the base set. The random jump to an arbitrary resource from the data graph is modeled by the vector e. A is the adjacency matrix which connects all available instances of the existing context ontology on one's desktop.

For creating the adjacency matrix A, the component makes use of the Beagle^{++} ontologies for extracting all metadata entries from the RDF store, created by the different Metadata Generator Modules. Being stored as triples, following the RDF subject-predicate-object model, the entries in the Sesame store provide the linkage information among the resources. The weights of the links between the instances correspond to the weights specified in the authority transfer schema graph. When instantiating the Beagle^{++}ontology for the resources existing on the users desktop, the corresponding matrix A will have elements which can either be 0, if there is no edge between the corresponding entities in the data graph, or have the value of the weight assigned to the edge in the authority transfer schema graph divided by the number of outgoing links of the same type. Once the matrix A is created, the rank computation is performed and the ObjectRank Computation Component writes the results back to the RDF Store. These new entries also follow the RDF subject-predicate-object model: the subject is represented by the URI of the resource whose ObjectRank is being stored, the predicate is a special property referring to the ObjectRank score, and the object is a float value representing the score itself.

Re-ranking Component. In contrast to the ObjectRank Computation Component, which runs as an independent application, the second component of the Ranking Module is integrated in Beagle. However, users have the possibility to choose one of the two ranking schemes: the one provided by the standard Beagle installation, based on TFxIDF measures, or the one we developed for Beagle^{++}, based on ObjectRank combined with TFxIDF. The first scheme is implicit. For the second one users have to start the Best^{++} client with an additional parameter.

The new ranking scheme we developed benefits both from the advantages of TFxIDF and those of ObjectRank. The new scores are computed as a combination of them using the following formula:

$$\mathrm{R}'(a) = \mathrm{R}(a) \cdot \mathrm{TFxIDF}(a), \tag{2}$$

where a represents the resource, $\mathrm{R}(a)$ is the computed ObjectRank, $\mathrm{TFxIDF}(a)$ is the TFxIDF score for resource a and $\mathrm{R}'(a)$ is the resulting score. The formula

guarantees that the highest ranked resources have both a high TFxIDF and a high ObjectRank score. The re-ranking is performed at query time, as we extract the ObjectRank values from the Sesame repository only for those resources which were returned by Beagle, i.e., those having a TFxIDF score greater than zero. Thus, the computation still delivers the output search results very fast.

3 Merging Metadata from Different Sources

Merging metadata from different sources means merging schema information as well as instance information. We will discuss our current solutions for both of these issues in this section. In the first subsection, we describe our Desktop Ontology, being developed in the context of the NEPOMUK project, which aims at providing an extensible ontology for desktop metadata. In the second subsection, we tackle the problem that metadata is provided by different sources, often with multiple identifiers for the same entity.

3.1 Desktop Ontology

In an environment such as Beagle^{++}, a large number of developers rely on the desktop ontology, and the functionalities of the environment need to be fully exploited in numerous and different domains. Second, part of the data present on our desktops is already—at least partly—annotated, for example by EXIF metadata[7] embedded in most of JPEG files. This implies that we need an easily extensible ontology which copes with existing metadata.

Based on these requirements we propose three ontology layers, inspired by [14]. Figure 3a shows the connections between these layers[8], in which an arrow indicates that an ontology makes references to the elements defined in the pointed ontology. Figure 3b is an example of the Publication class using elements defined in the three layers. We will briefly describe each of these layers.

The **Data Ontology**[9] models the metadata already present on our desktop, such as name and creation date for a "File", or more specific properties for subclasses of "File" (e.g., GPS coordinates for EXIF). Finally, beside all the existing types of desktop files, we also defined "Email" as a separate class, as it represents another important kind of desktop resource.

The **Desktop Ontology**[10] models the most common conceptual objects associated with the desktop items. It is centered around the classes "Person" and "Desktop_Document", while their sub-classes focus on more specific concepts such as "Image", "Sound", or "Text".

The **Domain Ontologies**[11] model *domain-dependent* objects. They represent specific functionalities in our desktop environment, or specific concepts for

[7] `http://www.exif.org/specifications.html`
[8] Part of this hierarchy is a result of joint discussions with Leo Sauermann from DFKI.
[9] `http://www.kbs.uni-hannover.de/beagle++/ontology/data\#`
[10] `http://www.kbs.uni-hannover.de/beagle++/ontology/desktop\#`
[11] `http://www.kbs.uni-hannover.de/beagle++/ontology/domain_l3s\#`

the user's institute or company. While we will usually have several domain ontologies, in our current context we rely on a single domain ontology with classes such as "Conference", "Wordnet_Term", or "Publication".

The use of layers gives two advantages: (1) controlled heterogeneity, as applications have a minimal set of common concepts to cooperate on (i.e., the Data and Desktop Ontologies), and (2) flexibility through the Domain Ontologies, in which additional metadata can be defined. We are currently implementing a new feature of the Metadata Storage Module, which will verify that every inserted RDF statement conforms to these ontologies, expressed in RDFS.

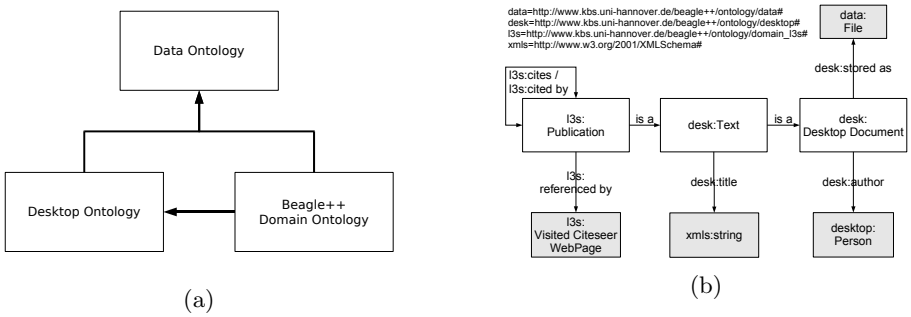

Fig. 3. (a) Beagle++ ontology hierarchy overview; (b) Specific subset of the Beagle++ ontology, focusing on Publications.

3.2 Entity Identification

Our Metadata Storage Module has to be able to merge metadata from different sources, which may refer to one entity using different identifiers. In our context, references to persons for example will be provided by different applications. A person can be the sender of an email, a co-author of a publication or even the co-author of a publication that is referred by your own publication. Each of these applications uses a different *method* to describe the specific person. For example, email address is used in the email context, full names in publications and the name's acronym in the reference publication. Since the main goal of merging all metadata is to enable us to coherently describe any desktop object, we need to discover all these different identifiers (i.e. email address, full name, and acronym) that refer/define a specific entity (i.e. person). The goal of Entity Identification is to discover all existing entities in a Metadata Storage Module and identify the objects which refer to these entities.

In our current Beagle^{++} environment, this Entity Identification problem is relatively limited because the metadata schema is rather small and fully known (Section 3.1). In this schema we identified the attributes which describe the same

entity, named *multi-purpose attributes*. Multi-purpose problematic attributes for the entity "Person" are "email_address" found in an email, "author" found in a publication, "creator" found in a document, and others. The Entity Identification algorithm is executed every time metadata is inserted into the Metadata Storage Module. We parse the new metadata to find whether they contain any multi-purpose attributes, and compare the value of these attributes with the value of the entities already stored in the repository. This comparison can give two conclusions. The first is that this object is a new entity and therefore it should be inserted as one. The second is that this object refers to an existing entity, such that no entry should be created for this entity, but only a link to show that this object refers to the specific entity. Comparison is done using a technique providing an approximation for the similarity between two strings, called *SecondString* [7].

The result of the Entity Identification algorithm is a set of cleaned metadata that describes the entities in a coherent way, which records for each entity the different references used to identify it. During search, Beagle^{++} uses this information to exploit all information about an entity regardless of the identifier used in its metadata.

4 Related Work

Several search and retrieval systems make extensive use of the semantical relationships that can be inferred on the desktop. Haystack [10] for example emphasizes the relationship between a particular person and her corpus. It creates RDF connections between documents with similar content and then exploits this information for facilitating browsing and information access. Unlike our work, their focus is on organizing the data, rather than searching it. Some basic search facilities came with Magnet [12], an additional Haystack component, yet relying on database querying approaches, which are complementary to ours.

The Gnowsis project [11] adds Semantic Web interfaces to common desktop applications such that documents are linked across applications and users are able to use their personal computer as a small personal semantic web. A number of adapters read data from different sources and make this information available as RDF. Created metadata is stored in a local RDF database and can be viewed through a browser. Additionally, this browser shows related information for resources and can be used for annotating photos or persons, for linking resources, as well as for full text search.

In [4, 5] we described our Beagle^{++} personal information system, a semantically enriched extension of the Beagle open source desktop search engine. We proposed various heuristics to generate ample activity based metadata associated to each desktop item. In addition, we generated links between resources in a similar manner to Haystack [10] and we applied a schema-based PageRank [2] to compute reputation scores. In this paper we do not focus on single innovative modules designed for a specific task, but rather present the toolbox architec-

ture and components which enables these modules to communicate and function properly.

Semantically enhanced search was also addressed from other perspectives, as in Stuff I've Seen [9], where contextual cues (e.g., access time, or author) are used to enrich search results, or as in Swoogle [8], in which information retrieval capabilities are offered for semantic documents residing on the Web.

Other algorithms focus more on the ranking scheme than on the semantically inferable connections on the desktop. Meza et al. [1] develop a ranking technique for the possible semantic associations between the entities of interest for a specific query. They define an ontology for describing the user's interest and use this information to compute weights for the links among the semantic entities. In our system, the user's interest is a consequence of her activities, this information being encapsulated in the properties of the entities defined, and the weights for the links being manually introduced.

An interesting technique for ranking the results of a query on the semantic web takes into consideration the inferencing processes that led to each result [13]. In this approach, the relevance of the returned results for a query is computed based upon the specificity of the relations (links) used when extracting information from the knowledge base. The calculation of the relevance is however a problem-sensitive decision, and therefore task oriented strategies should be developed for this computation.

Finally, the difficulty of accessing information on our computers has prompted several releases of desktop search applications recently, such as Google desktop search [12] (proprietary, for Windows) or the Beagle open source project for Linux[13]. Yet they only include already existing metadata in their system, such as email sender or file type, and otherwise focus on full-text search.

5 Conclusions and Further Work

In this paper we presented the architectural design details necessary for implementing a semantically enhanced desktop search application. We proposed a modular architecture and toolbox, in which components such as metadata generators and ranking calculators are easily integrated. Our current implementation of this toolbox builds upon a snapshot of the standard Beagle implementation, namely the version shipped with SUSE Linux 10.0. We provide a set of rpm-packages installable in SUSE 10.0 on our official Beagle^{++} web site[14], together with a ready-to-use virtual machine image for the VMware Player.

Motivated by the strong results Beagle^{++} illustrates, we are currently investigating several directions for improvements. First, we want to incorporate more advanced metadata generators and filters into the indexer, for example to cope with complex RDF path materialization and updating strategies. Second, we will generalize the suggested algorithms from Beagle^{++}'s specific environment

[12] `http://desktop.google.com/`
[13] `http://www.gnome.org/projects/beagle/`
[14] `http://beagle.l3s.de/`

(i.e., Linux) and provide them also for Windows based search implementations. Finally, we will focus on a Beagle^{++} version for supporting work groups and other social networks, and will include modules and interfaces for communication between distributed Beagle^{++} clients.

References

1. B. Aleman-Meza, C. Halaschek, I. B. Arpinar, and A. Sheth. Context-aware semantic association ranking. In *Semantic Web and Databases Workshop*, 2003.
2. A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *Intl. Conf. on Very Large Databases*, 2004.
3. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *Intl. Semantic Web Conf.*, 2002.
4. P. A. Chirita, R. Gavriloaie, S. Ghita, W. Nejdl, and R. Paiu. Activity based metadata for semantic desktop search. In *Proc. of the 2nd ESWC*, 2005.
5. P. A. Chirita, S. Ghita, W. Nejdl, and R. Paiu. Semantically enhanced searching and ranking on the desktop. In *Proc. of the Semantic Desktop Workshop held at the 4th Intl. Semantic Web Conf.*, 2005.
6. P. A. Chirita, S. Ghita, W. Nejdl, and R. Paiu. Beagle++: Semantically enhanced searching and ranking on the desktop. In *Proc. of the 3rd ESWC*, 2006.
7. W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Workshop on Inf. Integration on the Web*, 2003.
8. L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. C. Doshi, and J. Sachs. Swoogle: A search and metadata engine for the semantic web. In *Proc. of the 13th ACM Conf. on Information and Knowledge Management*, 2004.
9. S. Dumais, E. Cutrell, J. Cadiz, G. Jancke, R. Sarin, and D. C. Robbins. Stuff i've seen: A system for personal information retrieval and re-use. In *SIGIR*, 2003.
10. D. R. Karger, K. Bakshi, D. Huynh, D. Quan, and V. Sinha. Haystack: A customizable general-purpose information management tool for end users of semistructured data. In *Proc. of the 1st Intl. Conf. on Innovative Data Systems Research*, 2003.
11. L. Sauermann and S. Schwarz. Gnowsis adapter framework: Treating structured data sources as virtual rdf graphs. In *Intl. Semantic Web Conf.*, 2005.
12. V. Sinha and D. R. Karger. Magnet: supporting navigation in semistructured data environments. In *Proc. of the ACM SIGMOD Intl. Conf. on Mgmt. of Data*, 2005.
13. N. Stojanovic, R. Studer, and L. Stojanovic. An approach for the ranking of query results in the semantic web. In *Intl. Semantic Web Conf.*, 2003.
14. H. Xiao and I. F. Cruz. A multi-ontology approach for personal information management. In *Proc. of the 1st Workshop on The Semantic Desktop (ISWC)*, 2005.