FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Towards a distributed Concept Search Framework for Specialized Domains

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering and Internet Computing

eingereicht von

Christian H. Inzinger
Matrikelnummer 0225558

und

Johannes M. Schleicher
Matrikelnummer 0125876

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Univ.Prof. Mag.rer.soc.oec. Dr.rer.soc.oec. Schahram Dustdar
Mitwirkung: Dr. Hong-Linh Truong

Wien, 15.09.2010

(Unterschrift Verfasser) (Unterschrift Betreuer)

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Abstract

Specialized domains like Health Care, Space and Software Engineering rely on the efficient retrieval of digital objects in order to work effectively. Digital objects in these domains not only include typical text documents, but also multimedia objects, like presentations, audio and video elements. Furthermore, they are associated with domain specific concepts and persons, which are crucial factors in retrieving the desired information out of a vast number of possible candidates. The extensive amount of data, as well as the diverse types of objects, require advanced extraction and search mechanisms in order to enable efficient digital object retrieval. Although semantic extraction and retrieval approaches exist, a truly extensible framework, fully customizable to the domains' specifics, is missing.

In this thesis we present a distributed concept search framework for specialized domains that fills this gap. We introduce a combined semantic representations that allows the linking of domain specific concepts, keywords and social structures. Furthermore, we develop a highly adaptable and extensible software framework incorporating this combined representation, and demonstrate its feasibility with a prototype implementation.

Kurzfassung

Spezialisierte Domänen wie Gesundheitswesen, Raumfahrt und Softwareentwicklung verlassen sich auf effiziente Suchverfahren für digitale Objekte um effektiv arbeiten zu können. Digital Objekte in diesen Domänen umfassen nicht nur typische Textdokumente, sondern auch Multimediaobjekte wie Präsentationen, Videos und Audioelemente. Außerdem sind sie mit domänenspezifischen Konzepten und Personen verknüpft, die wesentliche Faktoren im Auffinden der gewünschten digitalen Objekte, aus unzähligen möglichen Kandidaten darstellen. Die beträchtliche Menge an Daten, sowie die unterschiedlichen Typen von Objekten, erfordern fortgeschrittene Extraktions- und Suchmechanismen um das effiziente Auffinden von digitalen Objekten zu ermöglichen. Obwohl semantische Extraktions- und Suchverfahren existieren, fehlt ein wirklich erweiterbares Framework, das spezifisch auf die Bedürfnisse der Domänen angepasst werden kann.

In dieser Diplomarbeit präsentieren wir ein Concept Search Framework für spezialisierte Domänen, das diese Lücke füllt. Wir stellen eine kombinierte semantische Repräsentation vor die es uns ermöglicht domänenspezifische Konzepte, Schlüsselworte und soziale Strukturen zu kombinieren. Weiters entwickeln wir ein anpassungsfähiges und erweiterbares Software Framework, welches sich dieser Repräsentation bedient und demonstrieren die Umsetzbarkeit mittels einer Prototyp Implementierung.

Acknowledgements

We thank Univ.Prof. Mag.rer.soc.oec. Dr.rer.soc.oec. Schahram Dustdar and Dr. Hong-Linh Truong for their supervision, support and patience.

We thank the ESA Team, in particular Andrei Aries, Andres Galvez and Tiago Soares, for their fruitful discussion and valuable inputs and feedback.

Last but not least we thank our families for their great support and incredible patience.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

Recent advancements in networks, storage, computing and mobile devices have enabled sheer volumes of data that need to be stored. Such data includes not only typical documents, like presentations and technical reports, but also multimedia objects like pictures, audio and video, as well associated metadata. This data is commonly referred to as Digital Objects. Due to the high volume of data and diverse types of objects, advanced extraction and search mechanism are mandatory in order to retrieve relevant digital objects. This is especially true for specialized domains like Health Care, Space, Law or Software Engineering where plain keyword search simply is insufficient for efficient search and retrieval [45]. Such domains require the digital objects to be associated with semantic concepts, mostly defined in Domain Specific Ontologies, as well as the association to certain social structures.

Traditional digital library systems [88; 62] were designed for generic digital objects, and provide no explicit considerations for the requirements of highly specialized domains, such as modeling of expertise or interest information, as well as specialized extraction and annotation mechanisms respecting formats prevalent in particular domains. In order to satisfy theses requirements, a combined representation of domain specific concepts, keywords, social structures, and the relations between them is necessary to enable data retrieval and analysis by domain knowledge. Beyond that, existing solutions do not provide means to allow for granular tailoring and distribution of system aspects like extraction and retrieval, but only offer predefined packages that may not suit the specific needs of the domain. In addition to that, the Web 2.0 has enabled the participation from users, thus valuable knowledge obtained from user participation and social networks could substantially improve domain specific information retrieval [56] and should be incorporated. Furthermore, the large amount of available data requires parallel algorithms and distributed components to handle the complex and voluminous data in specialized domains [31; 66].

1.2 Motivation

Although semantic extraction and retrieval mechanisms as well as distributed approaches exist for the scenario outlined above, a truly extensible framework is missing.

Current approaches revolve around static frameworks and systems that neither allow to taylor each part of the process to the requirements of the specific domain, nor easily enable the integration of new discoveries and algorithms for certain parts of the process.

Despite this, they miss a combined semantic representation that allows the connection of domain specific ontologies with social and keyword information.

1.3 Contribution

Our contribution is an extensible, distributed concept search framework overcoming the limitations outlined above.

We developed a flexible combined semantic representation and data model that allows the connection of different metadata representations, as well as domain specific ontologies (see Chapter 5), that can be utilized for different domains.

Utilizing this model as a fundament, we designed a distributed component based framework architecture, that allows the easy extensibility and adaptability of each part of the search process to the specific needs of the domain, as well as the integration of novel algorithms and methods at any time. Our expressive API further enables easy integration of existing systems and components, as well as the use of only specific parts of the framework that are viable for the specialized domain. Furthermore, we ensured distributability in order to support different optimization scenarios, like for example MapReduce/Hadoop, as well as a plethora of deployment scenarios.

Based on our design, we developed a Proof of Concept prototype. We used the Space Domain, within the context of the European Space Agency using ESA data, as specialized test domain for the prototype.

1.4 Organization of this Thesis

Part of this thesis has been published under the ESA Contract **ESA ITT Number RFQ 3-13016/09/NL/CBi**[55].

The rest of this thesis is organized as follows: Chapter 2 presents background information and terminologies for our thesis. Chapter 3 analyzes the state-of-the art of existing systems and relevant technologies. Chapter 4 presents the problem definition as well as use cases and requirements. Chapter 5 describes our proposed distributed social semantic search framework in detail. In Chapter 6, we present proof of concept demonstration of our framework. Chapter 7 gives a comprehensive comparison and concludes this thesis.

2 *Background*

The background for this thesis is in the domain of digital libraries [69; 83], in particular semantic digital libraries [24; 64], specifically viewed in a software framework perspective. Digital libraries research in general, and semantic digital libraries research in particular, have attracted much effort during the last few years. In the past, they have been mainly involved with the extraction of (semantic) information representing documents, the storage and management of the extracted information, and the search of documents based on that data. Therefore, as also mentioned in [1; 24], the relevant research topics are knowledge extraction (object recognition, segmentation and indexing, and semantic analysis), knowledge representation (document map, ontology), knowledge management (distributed or centralized storage), search algorithms, and knowledge visualization. Several techniques have been proposed for semantic digital libraries. Some have been developed for domain-specific digital libraries, such as life science literature search [33].

In this chapter we discuss background and terminologies which are necessary to fully understand the topic, and outline a traditional search process, as well as its differences to advanced search processes, which are relevant in a software framework perspective.

2.1 Typical Search Process

In order to explain relevant background and terminologies, we examine a simplified typical search process. Figure 2.1 shows a typical keyword based search process. The search process can be separated into three relevant phases.

The *first phase* is the **Query construction**. Traditionally, the user is given the ability to use Free Text Input, which can be limited to *keywords*, or in more advanced systems, the use of *natural language*. Additionally, the search process may provide Operators to augment the Text Input, like *boolean operators* or

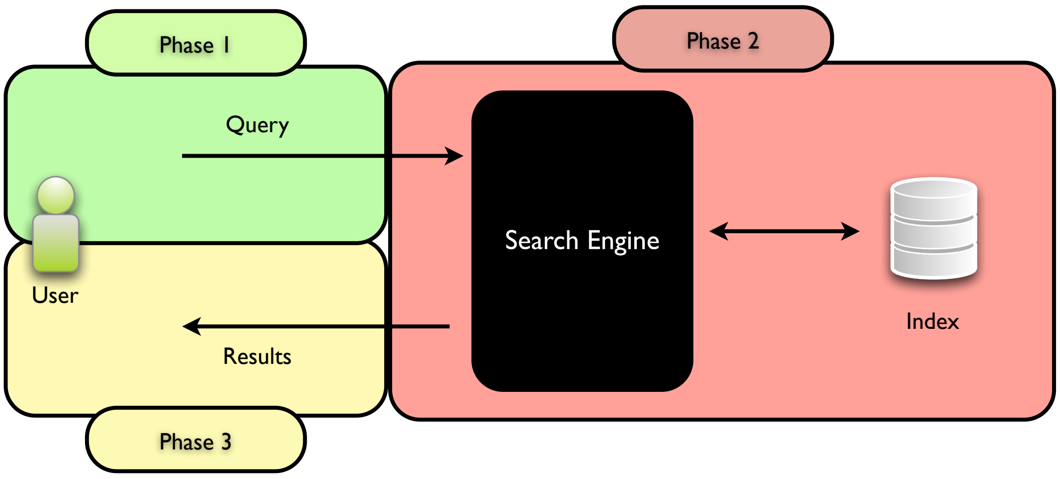


Figure 2.1: High level overview of a typical search process

regular expressions. Furthermore, Controlled Terms can be used to *disambiguate input*, *restrict output*, or *select predefined queries* from a value list or a faceted graph. Finally, the search process may provide early User feedback to facilitate *pre-query disambiguation* by means of a suggestion list or semantic autocompletion.

The user then submits the constructed query to the search engine, which starts the *second phase*. The search engine then evaluates the query and performs a **matching** procedure of the query against one or multiple indices of textual content and/or metadata, based on a search algorithm. The results of the performed matching are one or multiple digital objects.

Finally, the *third phase* is the presentation of the found digital objects to the user. The presentation phase **selects what to present** by providing means to constraint the result set. It further deals with **organizing the results** by clustering related objects, and finally orders them by relevance based on a ranking mechanism.

In the following section we will explain background and terminologies related to this search process.

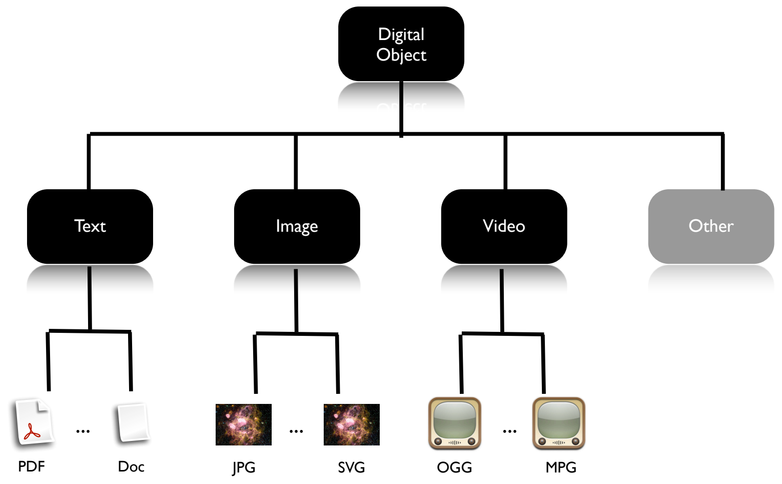


Figure 2.2: Digital Objects

2.2 Terminology

2.2.1 *Digital Object*

A *digital object* represents a discrete unit of information. It can be text, images, videos or any other digital representation of information. In the context of Digital Libraries a digital object can be defined as:

> *"An item as stored in a digital library, consisting of data, metadata, and an identifier."*[9]

In terms of access, we assume that each digital object can be uniquely distinguished using an identifier.

Digital objects can be represented in different formats (see Figure 2.2), and – depending on the specific format – different extraction mechanisms can be used to derive metadata from the digital objects. Conceptually, extracted metadata of a digital object can be embedded into or separated from the digital object. In our discussion, the extracted metadata is *not* part of the digital object, but a separate – albeit linked – representation, as seen in the following section.

2.2.2 *Metadata*

Metadata is data about a digital object. It characterizes the object it describes, and can represent different types of information about the digital object, such as contextual or provenance information. In literature, metadata types can be classified into two groups: (a) non-semantic and (b) semantic.

- **Non-semantic metadata**: *Non-semantic* metadata denotes the traditional form of descriptive metadata, such as keywords or a summary extracted from a text document.
- **Semantic metadata**: *Semantic* metadata enriches the information about a digital object using machine-comprehensible approaches, e.g. by associating ontological concepts with the digital object.

Metadata is one of the main sources which digital documents can be searched by. As this thesis is focused on semantic search, we will further outline relevant semantic representations of metadata.

Semantic Representations of Metadata

Semantic metadata can be represented using different forms. The main techniques to describe metadata in a semantic way are:

Categorizations via XML The most basic form of semantic representations is plain XML, with an according XML Schema, that defines the basic semantics. XML Schema is limited in a way that it only allows structuring of a document, without implicit support for document relations. The semantics of XML documents (e.g. attribute values, existence of child nodes etc.) can be queried using XQuery or XPath.

Taxonomies via RDF The Resource Description Framework (RDF)[1] is a metadata model. The basic expression in RDF is a triple consisting of *subject*, *predicate* and *object*. The *subject* denotes the resource, the *predicate* denotes aspects of the resource, and expresses a relationship between the *subject* and the *object*. The predominant query language is SPARQL[2], which is designed to query collections of triples, and easily traverse relationships. The ability to represent and traverse relationships based on RDF allows for greater expressibility than pure XML-based representations.

[1] http://www.w3.org/RDF/
[2] http://www.w3.org/TR/rdf-sparql-query/

Ontologies via OWL The Web Ontology Language (OWL)[3] is a set of knowledge representation languages for authoring ontologies. OWL is built atop RDF, with several additional features, such as relations between classes, cardinality, equality, more typing-of and characteristics-of properties, and enumerated classes. OWL further facilitates greater machine interpretability of web content than that supported by plain XML or RDF.

2.2.3 *Social Concepts and Social Semantic Combination*

Several search activities involve social interactions, such as finding an expert, getting recommendations from a colleagues, and manually generating metadata. The relevant background for searching with respect to social interaction lies on the concept of social networks, expert networks, and user participation.

Social Network

A social network [101] is a graph based social structure, consisting of individuals as nodes, and connections between these nodes. Each connection between nodes represents a relationship, which again can be associated with additional information.

Expert Network An expert network is a special form of a social network. Persons as nodes are being associated with certain topics, also represented as nodes, the relationship between persons and topics is then used to assess the expertise of a certain person for a specific topic.

Folksonomy

A folksonomy can be defined as the result of personal free tagging of digital objects for one's own retrieval. It can be further seen as the taxonomy-like structure, that emerges when large communities of users collectively tag digital objects [99; 89]. Figure 2.3 shows an example of a folksonomy. Folksonomies can be searched and queried using graph based mechanisms, and are limited in their accuracy to the expressibility of the tag. Different representations, like Tag Clouds [65], can be used to visualize the results.

[3] http://www.w3.org/TR/owl-features/

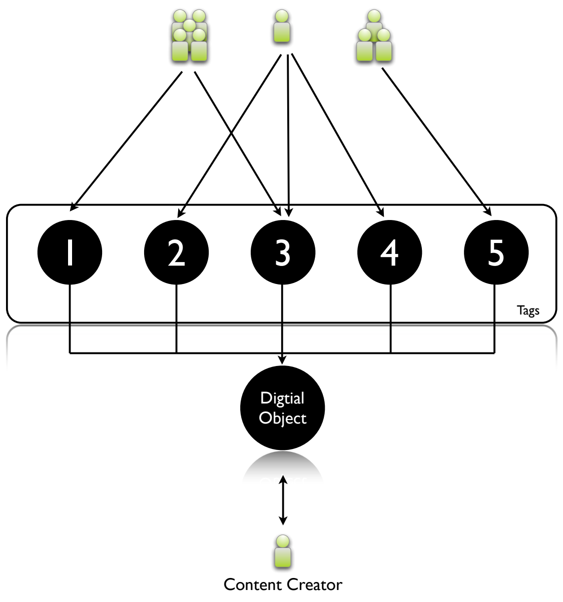


Figure 2.3: An example of a folksonomy

User Participation

User Participation is the process where a user actively augments the capability of an existing system [49; 93]. This can be the tagging related to folksonomies, or providing metadata for digital objects, as well as generating information for semantic extraction. To evaluate which user, or when a user should participate, social or expert networks can be used.

Social Semantic combination

A social semantic combination is the most advanced form of metadata representation. It is not only able to link document to document, or document to concept, or concept to concept, but introduces the ability to incorporate social structures (mentioned below), and link them to documents and concepts.

2.2.4 *Information retrieval systems*

Foundation for the search of digital objects are concepts in information retrieval systems, which are defined as

> *"Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)."* [71]

An information retrieval system can therefore be seen similar to a Digital Library. Information retrieval relies upon a prior information extraction, building the collection for the retrieval process.

Information Extraction

Information extraction (IE) can be defined as:

> *"IE may be seen as the activity of populating a structured information source (or database) from an unstructured, or free text, information source. This structured database is then used for some other purpose: for searching or analysis using conventional data- base queries or data-mining techniques; for generating a summary; for constructing indices into the source texts."* [44]

Semantic Extraction

Semantic extraction represents the enrichment of information extraction through semantic concepts like ontologies and/or social structures. Semantic extraction can be greatly improved, by utilizing user participation to overcome limitations of purely machine based approaches.

Indexing

Indexing is used to facilitate fast and accurate information retrieval, by building a compact representation of the digital object corpus.

2.2.5 Digital Libraries

The DELOS Digital Library Reference Model [11] has defined a Digital Library as follows:

> *"An organization, which might be virtual, that comprehensively collects, manages and preserves for the long term rich digital content, and offers to its user communities specialized functionality on that content, of measurable quality and according to codified policies."* [11]

Digital Library System

A digital library always incorporates a Digital Library System, which is defined as:

> *"A software system that is based on a defined (possibly distributed) architecture and provides all functionality required by a particular Digital Library. Users interact with a Digital Library through the corresponding Digital Library System." [11]*

According to this conceptual definition, a digital library fulfills two main functions:

- Management of digital objects, and
- Presentation of relevant digital objects to a consumer

Search and information retrieval activities are the core of the second function.

Classification of Digital Library Systems

Depending on how digital objects are managed and searched, different types of digital libraries can be discriminated. They can be classified based on the search mechanism (and as a consequence of the implementation of the search mechanism, there are differences in indexing, annotation, meta-data representation). The classification is not orthogonal, and the following digital library types can be derived:

- **Typical digital library** full-text index and keyword search.
- **Semantic digital library** digital objects are characterized by, indexed, and searched through semantic information[64].
- **Social-aware digital library** based on collaborative tags, and therefore a collaborative index, derived from social networks.

2.2.6 *Search Capabilities of Digital Libraries*

Search capabilities can be categorized into three main classes, namely keyword search, semantic search, and social search.

Keyword Search

The most basic form of search is keyword based [12]. In this form of search a keyword is matched against a text index of the document corpus.

Semantic Search

Semantic Search is widely used synonymical to concept-based search [91]. It seeks to improve search accuracy by understanding searcher intent and the contextual meaning of terms as they appear in the searchable data space, whether on the Web or within a closed system, to generate more relevant results.

Social Search

Social search is a type of search method, that determines the relevance of search results by considering the interactions or contributions of users utilizing social or expert networks [3]. When applied to search, this user-based approach to relevance is in contrast to established algorithmic or machine-based approaches, where relevance is determined by analyzing the text of each document, or the link structure of the documents.

2.2.7 *Result Representation*

The traditional presentation of retrieved digital objects, corresponding to a query, is organized as a ranked list. Several approaches have since been explored to extend the presentation of results, in order to convey more information, by displaying additional information or attributes for each of the retrieved documents, highlighting relationships between the documents and the query terms [50], visualizing inter-document similarities, or clustering related results [107].

3 *State of the Art of existing Systems*

In order to investigate how to complement and improve the results of domain-specific digital library search, we will examine existing digital library, information retrieval, and semantic search systems. The systems are chosen based on their relevance for highly specialized domains, that share the same background technologies.

The focus lies on the evaluation of the following characteristics, based on the Search Process, introduced in Section 2.1, from a systems perspective:

- The process starts with *Semantic extraction and annotation techniques*,
- followed by relevant *Storage mechanisms and concepts* for the extracted metadata.
- *Indexing techniques* are applied for efficient retrieval,
- *Search capabilities* cover basic search representations, as well as query construction;
- *Presentation of Search Results* concludes the functional characteristics, followed by
- *Non functional aspects* for search techniques, which will also be considered.

In the following sections, we will analyze existing approaches tackling the topics mentioned above. First, however, we will discuss existing metadata representations, as they are an important fundament of search processes in general and concept search in particular.

3.1 Metadata representations

3.1.1 Domain specific ontologies

For specialized domains several specific ontologies exists in this section we list some of the more prominent ones for different domains.

SWEET for the Space Domain

The most apparent metadata representation for the space domain are the SWEET[1] (Semantic Web for Earth and Environmental Terminology) ontologies, developed in NASA's Jet Propulsion Laboratory. The provided ontologies form a solid base-set of space-specific terminologies, and are suitable as semantic categorization of digital objects in a domain-specific digital library. However, the SWEET ontologies model *only* terminologies related to earth science data and information. Hence, they are not suitable to represent general metadata, such as document title, keywords, or creation date.

GALEN for the Medical Domain

GALEN is one of several medical ontologies. Its main intention was to enable a key element for architecture for interworking between medical record, decision support, information retrieval and natural language processing in healthcare [2]

SEOntology for the Software Engineering Domain

SEOntology is an example of an ontology for the Software Engineering Domain. It main intention is to enable a communication foundation for software engineering domain knowledge, by basically providing common software engineering concepts.[3]

3.1.2 Dublin Core

The Dublin Core Metadata element set and metadata terms [58] define canonical representations of common metadata attributes, such as title, author, creation/modification date, language and keywords. Hence, the Dublin Core

[1] http://sweet.jpl.nasa.gov
[2] http://www.openclinical.org/prj_galen.html
[3] http://www.seontology.org/

metadata elements are suitable for representing general metadata about a digital object. The representation of the author element, however, depends on yet another metadata framework aimed specifically at describing persons, as discussed in the following section.

3.1.3 FOAF

The FOAF[4] (Friend of a Friend) vocabulary [20] defines a metadata representation to describe people, companies, documents, and a basic set of relations between these concepts, such as name (of a person) or maker (of a document). The FOAF vocabulary is suitable for representing social structures, but it does not include specialized relations modeling notions such as expertise information, or granular topic relations between documents and topics.

3.1.4 SIOC

The Semantically Interlinked Online Community (SIOC) addressed the problem that online communities are islands of people and topics that are not interlinked. It's goal was to enable efficient information dissemination across communities by creating an ontology that models concepts identified in discussion methods [19].

3.1.5 SOAF

The Service of a Friend (SOAF) network proposed the integration of services into social networks. It enables the integration of services and humans into a common network structure and discusses the required extensions to existing social network vocabulary. The main benefits of the SOAF network are the ability to create a dynamic ecosystem of services, the ability to track relations between different stakeholders of web services as well as to provide information about service dependencies and input for creating dependency graphs of services [93].

[4]http://www.foaf-project.org

3.2 Semantic extraction and annotation techniques

To make content and context of digital objects explicit, several tools and techniques have been developed to generate and extract metadata, as surveyed in [95]. The generated metadata can either be directly associated to semantic concepts, or be used in later semantic extraction processes. In both cases, the ultimate goal of the generated metadata is to support the identification of digital objects.

3.2.1 *User participation*

Since not all kinds of metadata can be generated by automatic annotation mechanisms, manual annotation through user participation plays an important role in semantic extraction and annotation (see [95] for requirements on annotation by users and annotation tools). In order to avoid bottlenecks, that can be caused by excessive needs for manual annotation, it is crucial to provide simple and intuitive interfaces for annotation, that can easily incorporate existing information sources, and be integrated in activities of users [see 95; 98].

User participation is not only used to generate additional metadata by manual annotation, but also to provide information for semantic extraction. There are numerous systems relying heavily on user participation to annotate content and generate folksonomies, commonly known as "Web 2.0 applications", such as Flickr[5] and del.icio.us[6]. Several digital library systems also allow for tagging and annotation of digital objects [64].

Furthermore, in order to ensure minimal invasion in terms of user interruption, and also to make certain that user participation takes place, the principles of Luis v. Ahn [98] have shown to be feasible. Since time is limited for the user of the system, Captchas provide an interesting opportunity to augment the user participation process for semantic annotation.

3.2.2 *Automatic Semantic Extraction*

In addition to manual annotation discussed in Section 3.2.1, automation plays a very important role in the semantic extraction and annotation process. It

[5] http://flickr.com
[6] http://del.icio.us

is, therefore, important that automatic knowledge extraction technologies are integrated into the annotation process.

Standard formats

The use of standard formats will foster the interoperability and future extensions. Furthermore, they provide a very valuable bridging mechanism to share existing semantic information and annotations. To date, two popular types of standards used are OWL, for describing ontologies, and RDF, for annotations. For example, they are used in e.g. Fedora [88] and JeromeDL [64].

Ontology support

Usually, there are multiple ontologies dealing with different aspects of the domain. In addition to supporting standard formats, it is important that multiple ontologies are supported. Furthermore, as ontologies change over time, the changes should be supported in the extraction process in order to ensure that the extracted knowledge is up to date.

In the space domain, SWEET[7] is one example for a highly modular domain specific ontology with 4600 concepts in 150 modular ontologies, organized by subject.

Support for heterogenous document formats and document evolution

Traditionally *semantic web standards for annotation* tend to assume that documents only exist in either HTML or XML formats, therefore neglecting a lot of information stored in other formats. In order to be able to capture all relevant information for the domain it is necessary to support heterogeneous information types, like documents, images, audio and video as well as different formats for each of these types. Regardless of the plethora of formats it is also important to respect the fact that documents will change over time, therefore a new extraction might be necessary in order to provide sustainable accuracy for later retrieval.

[7] http://sweet.jpl.nasa.gov/ontology/

Relevant tools and algorithms

Several tools and algorithms have been developed for semantic extraction and annotation. Notable tool examples are KIM[8], Rainbow Project[9] and OntoMat[10]. A detailed survey analyzing existing tools can be found in [95].

3.3 Storage mechanisms and concepts

The utilized storage mechanisms and concepts are important in any search framework. Basically two storage models can be separated, namely the *word processor model* [see 95] where annotations are stored with the document and the *semantic web model* (see [see 95]) that assumes that annotations will be stored separately from the original. Depending on the chosen model different storage options can be applied.

When the *word processor model* is used, the document is implicitly linked to its metadata (as the metadata is stored with the document, and thus the document store also becomes the metadata store), but when the *semantic web model* is preferred, an explicit link between digital object and derived metadata must be established, i.e. by referencing the digital object's identifier (as specified in Section 2.2.1).

3.3.1 *Metadata storage*

As described in Chapter 2 metadata can be represented in many different formats, the standard format – as discussed in Section 2.2.2 – for semantic representations is RDF.

RDF Stores

RDF stores are storage solutions based on the concept of triple stores and provide facilities to efficiently store and retrieve RDF data. Despite this basic functionality they incorporate SPARQL, a SQL like query language to extract information from the graph based RDF model. This enables sophisticated queries on the extracted metadata beyond simple SQL queries, and therefore provides

[8] http://ontotext.com
[9] http://rainbow.vse.cz
[10] http://annotation.semanticweb.org/ontomat

the fundament for later semantic retrieval. The most relevant requirement for a RDF storage solution, besides non functional requirements like performance and scalability, is the coverage of SPARQL. The coverage of the SPARQL standard clearly influences the expressibility of queries and therefore power of the retrieval process.

Relevant Tools

The most common RDF storage solutions are Jena[11][74], Sesame[12][21], MPT-Store[13], and Mulgara[14]; a detailed evaluation of their performance can be found in [68].

3.3.2 *Digital object storage*

Digital objects, as described in Section 2.2.1, can be represented in different formats. Therefore, many different storage solutions exist. They can be separated into three main categories, namely *file system stores*, *relational stores*, and *document stores*.

The most flexible way to store digital objects is a document oriented storage facility. Document stores revolve around the concepts of self contained documents. This fits the *word processor model* described previously. The main characteristics of document stores are:

- Being schema free, e.g., objects of any structure can be stored
- Data is self-contained
- Efficient data retrieval
- Horizontal scalability

Relevant Tools

Notable examples are CouchDB[15] and MongoDB[16].

[11] http://jena.sourceforge.net
[12] http://www.openrdf.org
[13] http://mptstore.sourceforge.net
[14] http://www.mulgara.org
[15] http://couchdb.apache.org/
[16] http://mongodb.org/

3.4 Indexing Techniques

For efficient retrieval of digital objects, indices are required. Indexing approaches can be divided into three classes, *elementary indices*, *path look-up indices* and *navigational indices* [103].

3.4.1 *Elementary indices*

Elementary indices are the most basic form of index, and are based on tables. They ignore the notion of a path or higher concepts, their exclusive structural unit being the element or labelled node [103]. Examples for such elementary indices are basic text indices, but are of minor importance for semantic search, although they provide a necessary fundament.

3.4.2 *Path look-up indices*

Opposed to elementary indices, path look-up indices use entire document paths, instead of nodes as their basic structural unit. They mainly differ from elementary indices by storing each node's structural context, therefore enabling a solid fundament for semantic search. However, they are limited in a way, that only atomic lookups are possible, meaning they only retrieve the label paths, and possibly the IDs of all nodes where a given keyword occurs [103].

3.4.3 *Navigational indices*

Navigational indices provide the most sophisticated index representation, by using directed graphs as their main data structure, therefore representing the optimal indexing technique for RDF based metadata.

3.4.4 *Relevant Algorithms and Tools*

Notable algorithms are BitCube [106] (path-lookup), BUS [86] (navigational) or T-Index [108] (navigational), for a detailed explanation and comparison see [103].

Notable tools for indexing are Lucene[17], Lemur[18], and Terrier[19]; the tools are standard full text indexing solutions, and provide no specific support for domain requirements, beyond full text indexing.

3.5 Search Capabilities in Digital Libraries

From a systems perspective, there are multiple search mechanisms. They are non exclusive, and should be combined in an optimal scenario.

3.5.1 *Keyword based search*

Keyword based searches are the most basic form of search representations, backed by a full text index. Keyword based search represents a quasi-standard, and is part of every search system.

Special Domain Specific Keyword Search Systems Notable examples of keyword based search systems are IDS and ESTree.

Query construction in keyword based searches

Traditionally, the system is able to process free text input, this can either be limited to *keywords*, or, in more advanced systems, the use of *natural language*. Beyond this basic form of query construction the system can provide several operators like *boolean operators* or *regular expression* to enhance the express ability of the constructed query.

Matching in keyword based searches

Keyword based search uses syntactic matching of the query against a text index (see Section 3.4.1)

3.5.2 *Semantic – Ontology-based Search/Facet Search*

Facet search and ontology-based search are basically identical. Faceted search, also called faceted browsing, allows the user to explore information by *faceted*

[17]http://lucene.apache.org
[18]http://www.lemurproject.org
[19]http://terrier.org

classification, which allows the assignment of multiple classifications to an object, enabling the classifications to be ordered in multiple ways, rather than a single taxonomic order [40]. Faceted search allows users to navigate a multidimensional information space by combining basic keyword search with a progressive narrowing of choices in each dimension. Notable realizations of this concept are, for example MultiBeeBrowse [63]

Query construction in semantic searches

Query construction for semantic searches can be similar to keyword based searches, by providing the facilities mentioned in Section 3.5.1, to identify facets and combine facets. Despite this, there are graphical approaches like TagClouds [65] and TileBars [12, p.292], that simplify the query construction.

Matching in semantic searches

Semantic search uses semantic matching in addition to syntactic matching. It is used to further constrain or modify the result set by either utilizing graph traversal, explicit use of thesauri relations and inferencing based on the formal semantics of RDF, RDFS and OWL [54].

3.5.3 *Social – Community enabled browsing*

Community enabled browsing is based on the concept of folksonomies. It augments keyword based search approaches by utilizing social semantic concepts; essentially the user is enabled to consume the knowledge collected by entities in a community (e.g. a social network). A prominent algorithm to provide this feature is Social Semantic Collaborative Filtering (SSCF) [64].

Query construction in social searches

There is to the best of our knowledge currently no specific consideration of social aspects in query construction, as the mechanism introduced in Section 3.5.1 and Section 3.5.2 have shown to be sufficient.

Matching in social searches

Social search uses semantic matching shown in Section 3.5.2, based on the generated folksonomies represented as a graph, taxonomy or ontology. A notable example is del.icio.us[20].

3.6 Result presentation and Feedback for Search and Search Results

The basic result presentation is a list of eligible candidates matching the search query. Each list item consists of a *link to the retrieved digital object*, and is ordered according to some *score/rank*. For ranking results, several algorithms have emerged. However, the most prominent ones, which were adapted for semantic and social searches, are *PageRank*[78] and *HITS*[61]. The visualization for result presentations range from the most simple form as *text*, over *2-dimensional*, to *3-dimensional* representations.

Additionally, supported interfaces and devices play a valuable role. Currently, search can be conducted through *webrowser*, *rich native client interfaces*, and *mobile devices*, like smart-phones and tablet devices. In addition to human-centric interfaces, machine query able interfaces for robots are of relevance as well. Table 3.1 shows an overview of existing interfaces and representations.

Actor	**Interface**	**Representations**
Human	Web	HTML, XHTML
	Rich native	WPF[21], Cocoa[22], Quartz[23], Qt[24]
	Mobile	WML, Cocoa Touch[25], Android[26]
Systems/ Robots	SOAP	XML
	REST[42]	JSON[27]
	XML-RPC	XML

Table 3.1: Interfaces and used representations

[20] http://del.icio.us
[21] http://windowsclient.net/wpf/
[22] http://developer.apple.com/cocoa/
[23] http://developer.apple.com/graphicsimaging/quartz/quartzcomposer.html
[24] http://qt.nokia.com/
[25] http://developer.apple.com/iphone/
[26] http://www.android.com/
[27] http://www.json.org/

3.7 Non-functional Aspects for Search Techniques

Beside functional aspects discussed above, it is also relevant how the search techniques assess non-functional aspects, e.g. scalability, parallelization of index generation and querying, as well as their suitability for distributed computing, since we are always concerned with distributed multi-user systems.

Index	**Storage**	**Matters of concern**
One	One	Storage will eventually be the bottleneck
One/Sharded	Multiple	Popular sharded index partition will eventually be the bottleneck
One/Replicated	Multiple	Memory intensive, complex update procedure
Multiple	Multiple	The more heterogeneous the environment the higher the complexity

Table 3.2: Index/Storage scenarios

Organization model	**Running system**	**Description**
Centralized	Server	Single Server
	Cluster system	Computing cluster with single point of entry
Distributed	P2P system	Multiple distinct sites able to interoperate
	Cloud system	Multiple sites able to adjust to varying loads

Table 3.3: Distribution scenarios

One of the main non-functional aspects of digital library systems is the handling of distributed setups and parallel query and information extraction processing, as shown in Table 3.2 and Table 3.3. In order to address the matters of concern outlined in Table 3.2, several approaches have emerged. For example, MapReduce was developed to handle the computations that process large amounts of data, such as crawled documents [32]. Therefore, it represents a very valuable concept for optimized indexing and extraction in information retrieval and processing, and is also suitable for semantic annotations [66].

3.8 Relevant Reference Architectures

So far we only are aware of DELOS which is the EU Network of Excellence in Digital Libraries and helps to conduct and share results of research on next generation technologies for digital libraries[28]. DELOS aims at delivering research and prototypes of DL technologies using P2P, Grid and SOA. The key deliverable of DELOS is a reference model for (future) Digital Library Management Systems. Currently, DELOS provides a stable prototype of Digital Library Management System (DLMS) representing the digital library management part of the proposed reference architecture.

Delos is relevant as a reference architecture for Digital Library Solutions. The main concepts introduced by the DELOS Digital Library Manifesto build upon six orthogonal and complementary domains that together characterize the Digital Library universe, these are:

- *Content*, representing the information made available
- *User*, representing the actors interacting with the system
- *Functionality*, representing the facilities supported
- *Policy*, representing the rules and conditions
- *Quality* and
- *Architecture*

Furthermore, it introduces different types of actors, and states that the ultimate goal of the whole reference model activity is to clarify the Digital Library universe to the different actors, by tailoring the representation to their specific needs. Specific details can be found in [11].

[28] http://delos.info

3.9 Existing Systems as a Whole

3.9.1 FEDORA

The Flexible Extensible Digital Object Repository Architecture[29] (FEDORA) enables a wide array of different applications, like scholarly and scientific workbenches, data curation, linking and publishing, as well as collaborative repositories and integrated knowledge spaces. It is based on a flexible digital object model that can support documents, images, as well as complex multimedia publications, and enables the creation of so called networks of objects using RDF.

FEDORA provides a natural model for exposing repositories as a network of objects, indexing based on a generalizable data model and extensible enrichment of object descriptions. A FEDORA repository provides the following features:

- Generic Digital Object Model
- Automatic content versioning and audit trail
- Web Service Interfaces (REST and SOAP)
- RDF based storage.

3.9.2 BRICKS

BRICKS[30] aims at establishing the organizational and technological foundations for a digital library network in the cultural heritage domain. It provides

- A software infrastructure for building digital library networks
- A set of end-user applications
- A business model

From the architecture point of view, BRICKS has the following key characteristics:

- A decentralized P2P network

[29] http://www.fedora-commons.org/
[30] http://www.brickscommunity.org

- Each P2P node is a set of SOA components as well as
- components for storing, accessing, searching and browsing digital objects.

3.9.3 *JeromeDL*

JeromeDL is – to the best of our knowledge – the only social semantic digital library system available [64]. The main motivations of JeromeDL were the integration and search of information from different bibliographical sources, as well as to share and interconnect knowledge among people. Its main features are

- interconnection of meaningful resource description of with social media
- enhanced personalized search facility
- integrated social networking with user profiling
- extensible access control based on social networks
- collaborative browsing and filtering
- dynamic collections
- integration with Web 2.0 services

3.9.4 *Overview of the systems above*

Table 3.4 summarizes features of the above-mentioned systems.

System	**Semantic Extraction and Annotation**	**Storage mechanism and concepts**	**Distribution techniques**	**Search Capabilities**
Fedora	Own Middleware component for annotations	RDBMS/ Mulgara	federation via name resolver search services; P2P	Field Search, Ontology-based, Full-Text
BRICKS	Own Middleware component for annotations	Jena compliant backend	Fully decentralized (P2P)	Full-text, Field-Search, Ontology-based
JeromeDL	Free tagging taxonomy based (JOnto), SSCF	Sesame compliant backend	Distributed searching (P2P), aggregated browsing (hierarchical)	Full-text, Field-Search, Ontology-based, NL Query Templates

Table 3.4: Overview of digital library systems

4 Problem Definition

In this chapter, we will discuss the problem to be tackled, justify that it has not previously been answered, and state why it is worthwhile to provide a solution for it.

4.1 Research Question and Approach

The discussion of the State of the Art of existing Systems in Chapter 3 clearly shows that a distributed concept search framework, allowing to taylor each part of the process to the requirements of the specific domain, as well as easily enable the integration of new discoveries and algorithms for certain parts of the process does not exist as of yet. Despite this, a combined semantic representation that allows the connection of domain specific ontologies with social and keyword is missing. Specifically shortcomings concerning the following key factors have not been properly addressed:

- Combined social semantic representation able to incorporate any domain specific ontology
- Design for easy adaptability of specific components
- Design for granular distributed computing
- Integration of advanced user interfaces, visualizations and user participation means

As presented in Section 3.1, there are several approaches to metadata representation, each suitable for different areas or domains. To the best of our knowledge, however, a combined representation of social semantic metadata, enriching basic possessive associations by allowing the modeling of additional information, such as expertise, and integrating any domain specific ontology does not yet exist.

The systems presented in Section 3.9 were designed to serve as digital library systems for generic digital objects, and provide no explicit considerations for the requirements of highly specialized domains, such as modeling of expertise or interest information, as well as specialized extraction and annotation mechanisms respecting formats prevalent in particular domains.

In the discussion of the state of the art we present several approaches to semantic extraction and annotation (Section 3.2), storage mechanisms and concepts (Section 3.3), and indexing techniques (Section 3.4) that are reasonably well suited for distributed computing. However, existing solutions do not provide means that allow for granular distribution of system aspects over processing resources, but only offer ways to scale the solution by means of replicating the whole system (with some allowing for the separate distribution of the storage and/or indexing component as a whole) on multiple machines to facilitate higher performance, scalability, and availability.

User interfaces of existing systems (see Section 3.9) are mostly realized as more or less complex input forms [88; 62], representing the internal metadata model, resulting in mediocre user experience at best, when trying to peruse systems beyond basic keyword based search. In the field of digital library systems, there is little research on improving the quality of user interaction to facilitate complex queries and analysis tasks in an intuitive manner (see Section 3.5). We will enable the easy dynamic integration of advanced Interface mechanisms such as MultiBeeBrowse [63], tag clouds [65] or TileBars [12, p.292] to assist users in intuitively constructing queries against the system, as well as analyzing retrieved data sets.

— ❧ —

We will develop a distributed, concept search framework for specialized domains incorporating an extensible combined semantic representation addressing the shortcomings outlined above. Our goal is to create an easily extendable and adaptable component architecture for social semantic extraction and search, that allows for each and every component to be exchanged. Furthermore, we will focus on the ability to easily distribute the system's component to facilitate high performance, scalability, and availability.

— ❧ —

In order to develop a distributed concept search framework for specialized domains we will first identify common use cases and requirements. Based on them we will develop an extensible combined semantic representation and data model that will serve as the foundation for our framework. The key point for this representation is the ability to integrate multiple semantic representations, specifically different domain specific ontologies, which will enable a truly adaptable concept search framework. After that we will design a distributed component based architecture that allows the simple adaption and extensibility of each component in order to incorporate different specific solutions for certain areas of the framework. This will ensure that certain parts of the framework can be exchanged and evolve in order to adapt to the different requirements for each domain.

4.2 Use Cases

In order to understand requirements, we focus on some common use cases, which are relevant to the search activities in specialized domains.

4.2.1 *Search Related Documents during Concurrent Domain Activity*

Description

In the case of a concurrent domain activity, each participating party is responsible for a different part of the process. In this process, the person responsible (user) frequently consults the search system to find relevant documents related to the task at hand, as well as experts related to the topic, and/or examined documents that could be consulted in case further questions emerge. The user searches the system by either specifying keywords or topic names to query the system. Additionally, they can utilize the domain specific ontology (DSO) to browse by topic, and retrieve related documents, as well as to constrain the browsing result with additional keywords. During the retrieval process the user can also tag retrieved documents with existing topics from the DSO or define new tags that augment the DSO.

Goal

Retrieving relevant documents, matching the specified keywords, and/or selected topics from the domain ontology ordered by relevance, as well as the associated experts/contributing persons.

Basic course of events

Figure 4.1 shows the activity diagram for the use case.

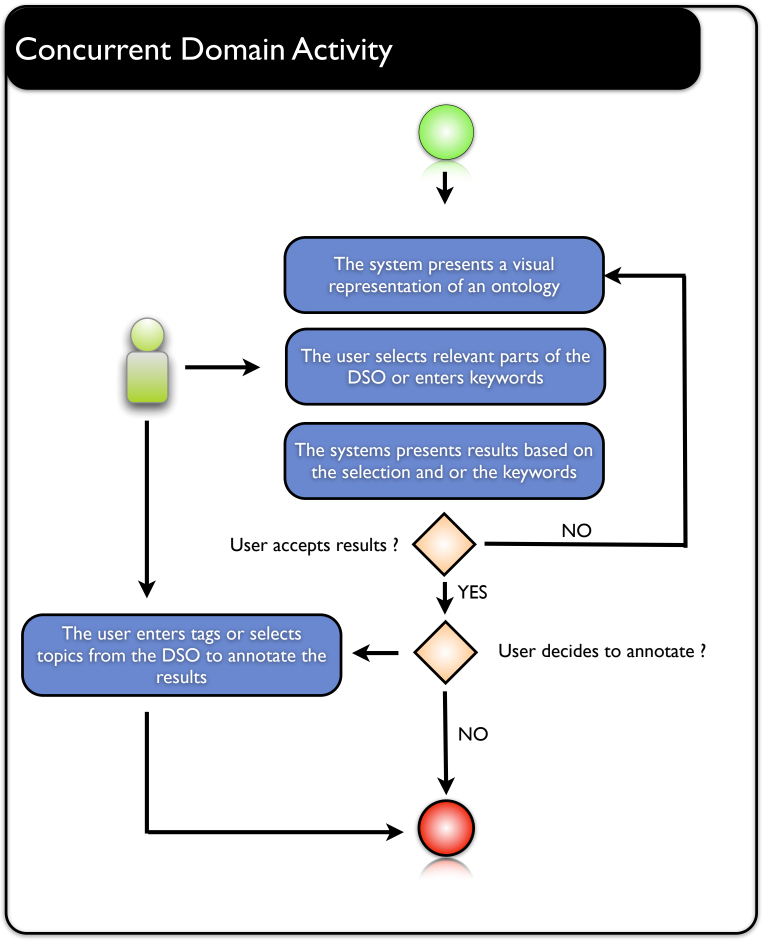

Concurrent Domain Activity
The system presents a visual representation of an ontology
The user selects relevant parts of the DSO or enters keywords
The systems presents results based on the selection and or the keywords
User accepts results ?
NO
YES
The user enters tags or selects topics from the DSO to annotate the results
User decides to annotate ?
NO

Figure 4.1: Activity Diagram for Concurrent Domain Activity

4.2.2 *Expert based Digital Object Retrieval*

Description

In the case of expert based retrieval, the user knows which persons could provide information relevant for his current research topic. He wants to retrieve digital objects that are related to his topic of interest, in conjunction with the persons that will provide the most relevant results. The user searches the system by either specifying keywords or topic names, as well as relevant persons to constrain the results. During the retrieval process, the user can also annotate if certain results provided by a person were relevant to the topic or not.

Goal

Retrieving relevant documents matching the specified topic and selected experts for the topic.

Basic course of events

Figure 4.2 shows the activity diagram for the use case.

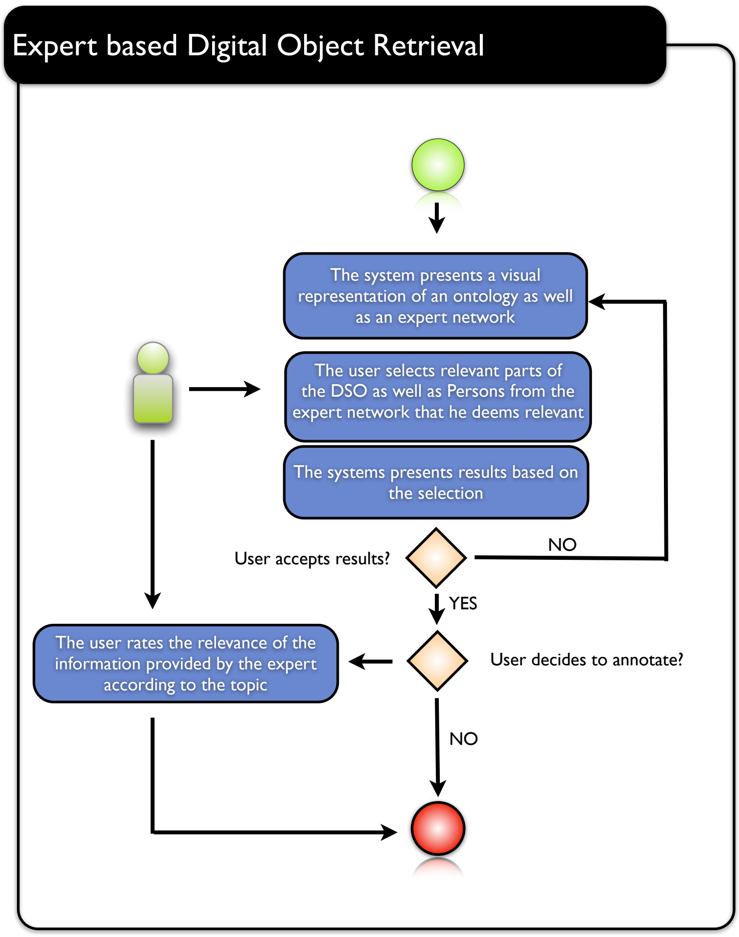

Expert based Digital Object Retrieval
The system presents a visual representation of an ontology as well as an expert network
The user selects relevant parts of the DSO as well as Persons from the expert network that he deems relevant
The systems presents results based on the selection
User accepts results?
NO
YES
User decides to annotate?
The user rates the relevance of the information provided by the expert according to the topic
NO

Figure 4.2: Activity Diagram for Expert based Digital Object Retrieval

4.2.3 *Media Library search*

In the case of media library search, the user participated, or is interested in a presentation held in, or related to the specialized domain. He knows certain keywords or persons related to the presentation, and/or images or videos he has seen. He wants to retrieve digital objects, that are related to the specified keywords, or the person that held the presentation. During the retrieval process, the user can also annotate and rank the retrieved digital objects.

Description

Goal

Retrieving relevant videos, images and presentations matching the specified topic/keywords and/or specified persons.

Basic course of events

Figure 4.3 shows the activity diagram for the use case.

4.3 Requirements

The requirements are being derived from our research in Chapter 3, in alignment with the Use Cases and Research Question mentioned above. They cover the areas of functional, non-functional and software engineering requirements, and provide the fundament for our proposed social semantic digital library framework.

- **high**: must be supported.
- **medium**: should be supported.
- **low**: not required but nice to have.

4.3.1 *Functional Requirements*

Functional requirements specify requirements related digital objects, metadata, semantic extraction and annotation, search techniques, user interaction and result presentation.

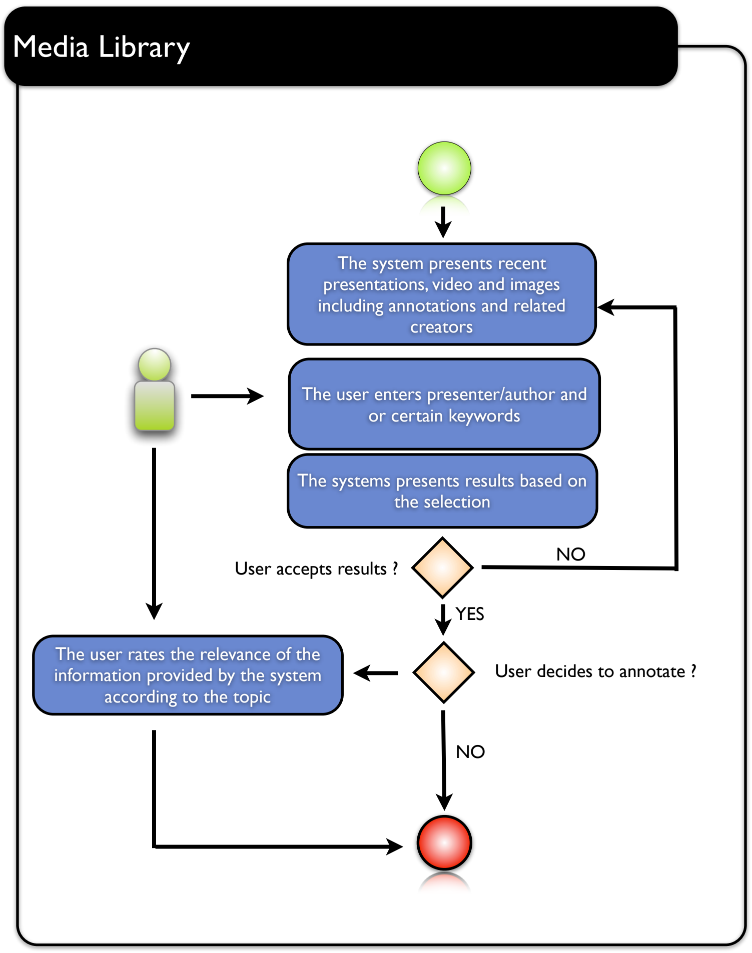

Media Library
The system presents recent presentations, video and images including annotations and related creators
The user enters presenter/author and or certain keywords
The systems presents results based on the selection
User accepts results ?
NO
YES
User decides to annotate ?
The user rates the relevance of the information provided by the system according to the topic
NO

Figure 4.3: Activity Diagram for Media Library

Supported Digital Objects

Is concerned which formats and types are supported as possible input and resource for the digital library. Table 4.1 shows the basic requirements for supported digital objects. Table 4.2 shows the requirements for digital object formats.

Type	Requirement
Document	High
Video	High
Image	High
Audio	Low

Table 4.1: Supported Digital Objects

Type	Format	Requirement
Document	PDF	High
	Powerpoint	High
	Word	High
	Excel	High
Image	PNG	High
	JPG	Medium
	TIFF	Low
Video	mov	High
	h.264	Medium
	mpeg2	Low
Audio	wav	Medium
	mp3	High
	ogg	Low

Table 4.2: Supported Digital Object Formats

Supported Extraction Mechanisms

The supported extraction mechanisms play a valuable role for ESA. Beyond basic text extraction with all its flavors like stemming etc., which is a basic requirement, it is necessary to regard the following requirements for semantic extraction shown in Table 4.3.

Criteria	Sub-criteria	Requirement
Information Representation	Social Semantic Representation	High
Supported Standard formats	RDF	High
	OWL	High
User participation	Incremental concept build up	High
Annotation Storage	word processor model	Medium
	semantic web model	High
Ontology support for metadata	Multiple Ontology support	High
Incorporation of existing Digital Object stores		High

Table 4.3: Requirements for Semantic Extraction Mechanisms

Supported Annotation Mechanisms

In addition to the supported extraction mechanisms, the following annotation requirements need to be respected as shown in Table 4.4.

Criteria	Sub-criteria	Requirement
User Participation	Tagging	High
	Rating of results and semantic or social annotations	High
Annotation Storage	word processor model	Medium
	semantic web model	High
Incorporation of existing systems for annotation		High

Table 4.4: Requirements for Semantic Annotation Mechanisms

Supported Indices

In order to enable social semantic search and incorporate existing indices in a minimal invasive way, certain requirements for supported indices have to be met.

Criteria	Sub-criteria	Requirement
Supported index types	Elementary Indices	Medium
	Path look-up indices	High
	Navigational indices	High
Incorporation of existing indices	via direct access to database	Low
	via service based middle layer	High
Distribution and Recovery	Distributed Indices	Medium
	Ability to restore broken indices	High

Table 4.5: Requirements for Indices

Supported Search Capabilities

Despite the basic requirements for keyword based search, social and semantic search must also be taken into account, see Table 4.6.

Criteria	**Sub-criteria**	**Sub-sub-criteria**	**Requirement**
Keyword based	Query construction	Natural Language	Medium
		Boolean Operators	Medium
		Regular Expressions	Low
Semantic Search	Faceted Browsing	-	High
	Ontology Search and Browsing	-	High
Social Search	Expert Network support	The ability to search for experts	High
	Folksonomy Support	-	High
	Social Collaborative Filtering	-	High
Combined Search	Semantically enriched keyword search	-	High
	Socially enriched keyword search	-	High
	Socially enriched semantic browsing	-	High

Table 4.6: Requirements for Search capabilities

Interfaces

Different requirements must be met, depending on whether a human or a machine interfaces the system, as shown in Table 4.7.

Criteria	Sub-criteria	Requirement
HCI	Web Interface	High
	Native Client	Low
	Mobile Client	Medium
Automation	SOAP	Medium
	REST	High
	XML-RPC	Low

Table 4.7: Requirements for Interfaces and used representations

4.3.2 *Non-functional Requirements*

The Non-functional Requirements are concerned with the system's capabilities in terms of reusability, extensibility and interoperability, as shown in Table 4.8.

Criteria	Sub-criteria	Requirement
Reusability	Integration of existing systems and components into the solutions	High
	Service-oriented	High
Extensibility	The system should provide API access to its functionality	High
	Modular architecture for easy functional extensions	High
Interoperability	Ability to access and integrate web based legacy systems	High

Continued on next page

Table 4.8 – continued from previous page

Criteria	**Sub-criteria**	**Requirement**
	Ability to incorporate existing authentication solutions	High

Table 4.8: Engineering requirements

5 Specification of a distributed Concept Search Framework for Specialized Domains

In this chapter, we present the architecture of a concept search framework for specialized domains, based on the requirements and research question specified in Chapter 4. We discuss a feasible implementation and an evaluation roadmap for the proposed framework.

5.1 Linked Model for Representing Information in the Concept Search Framework

In order to enable concept search for specialized domains, where the search is related to not only digital objects, but also experts and domain specific ontologies, it is necessary to understand the influencing factors on a digital object. A digital object searched and retrieved by the user is associated with keywords as well as semantic concepts, and is further authored or known by several other persons. Figure 5.1 illustrates this relation. These influencing factors must be described in a linked model, so that the concept search can be performed.

Despite that concepts might be related to each other as well, expressing the fact that they are associated or derived semantic representations. The last relevant factor is the association between persons which is commonly known as a social network.

For most of these associations, semantic representations already exist, however, what is missing is a way to link them together, in order to represent a digital object in the desired way, suitable for the concept search framework in specialized domains.

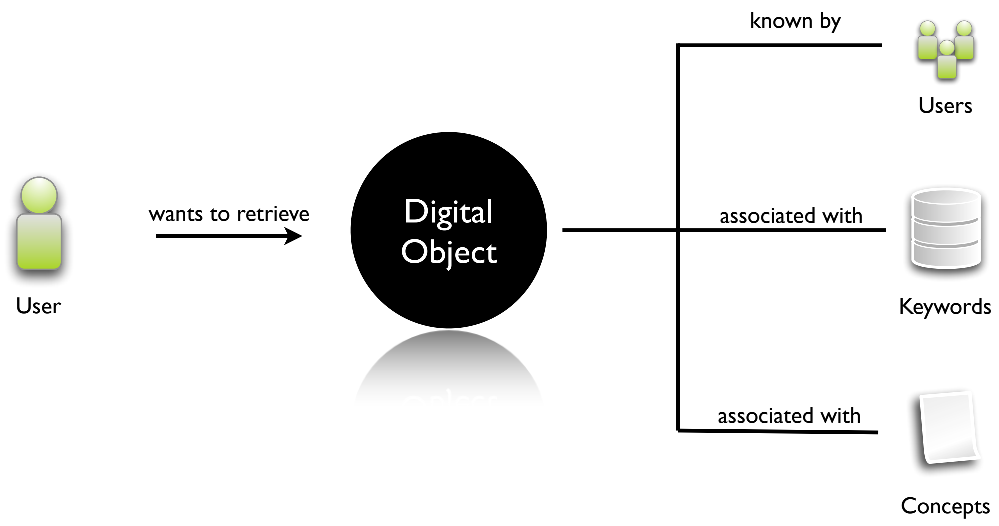


Figure 5.1: Influencing factors

In order to cover relevant representations, as seen above, for digital objects, we need to develop a semantic representation that is able to link all relevant factors. Figure 5.2 shows the high-level view of our linked model.

Because several different semantic representations have been developed, in our work, we utilize the following representations to generate our linked model:

- **FOAF**: to model basic social relations and personal information
- **DublinCore**: providing a base set of metadata
- **Thing**: to incorporate domain-specific concepts, we refer to Thing and therefore are able to incorporate any Domain Specific Ontology like SWEET, GALEN or SEOntology etc..

By integrating domain specific ontologies via Thing we ensure that our representation is extensible to any specific domain. Thing is the common base class of all OWL classes and therefore allows us to integrate any domain specific ontology. It further allows us to utilize certain subsets of the ontology (since they are all derived from Thing) or the whole ontology and therefore enables an easily adaptable integration of domain specific ontologies for specialized domains. In the following, we will define the data model linking the mentioned representations.

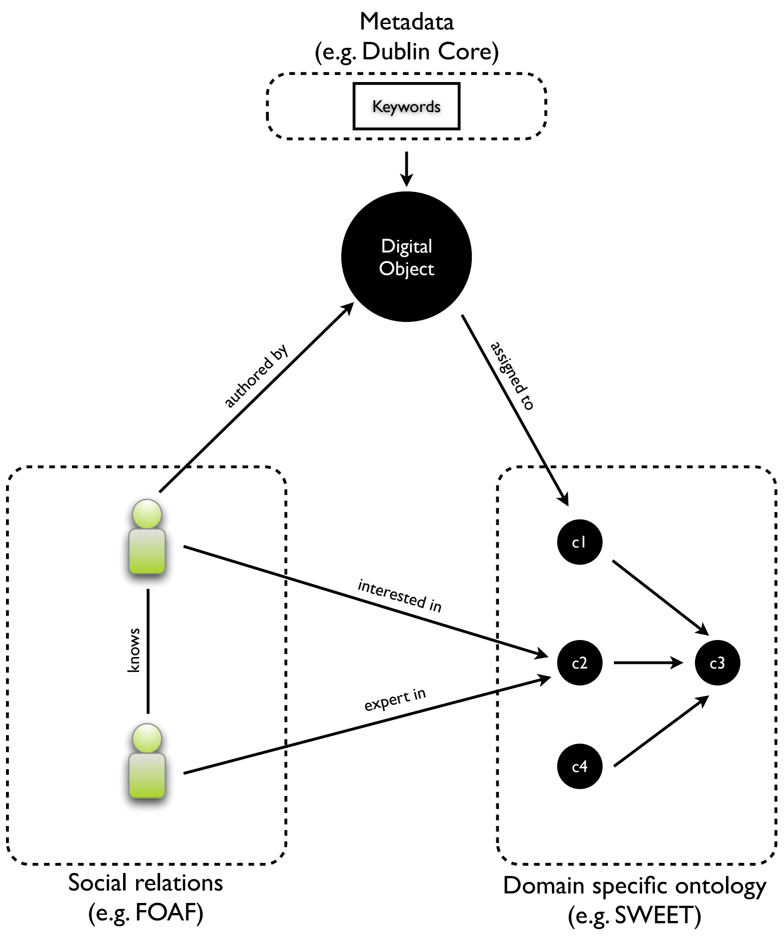


Figure 5.2: Semantic Representation

5.1.1 *Data model*

As existing representations do not cover all of the semantic information needed for the realization of our approach, we propose an ontology, defining classes and relations augmenting current approaches. Figure 5.3 shows a graphical overview of the created ontology.

Specifically, we defined the following classes:

- `Person`: was introduced to model persons in specialized domains.
- `Group`: represents teams in specialized domains.
- `DigitalObject`: is the basic unit of information for the semantic digital library.

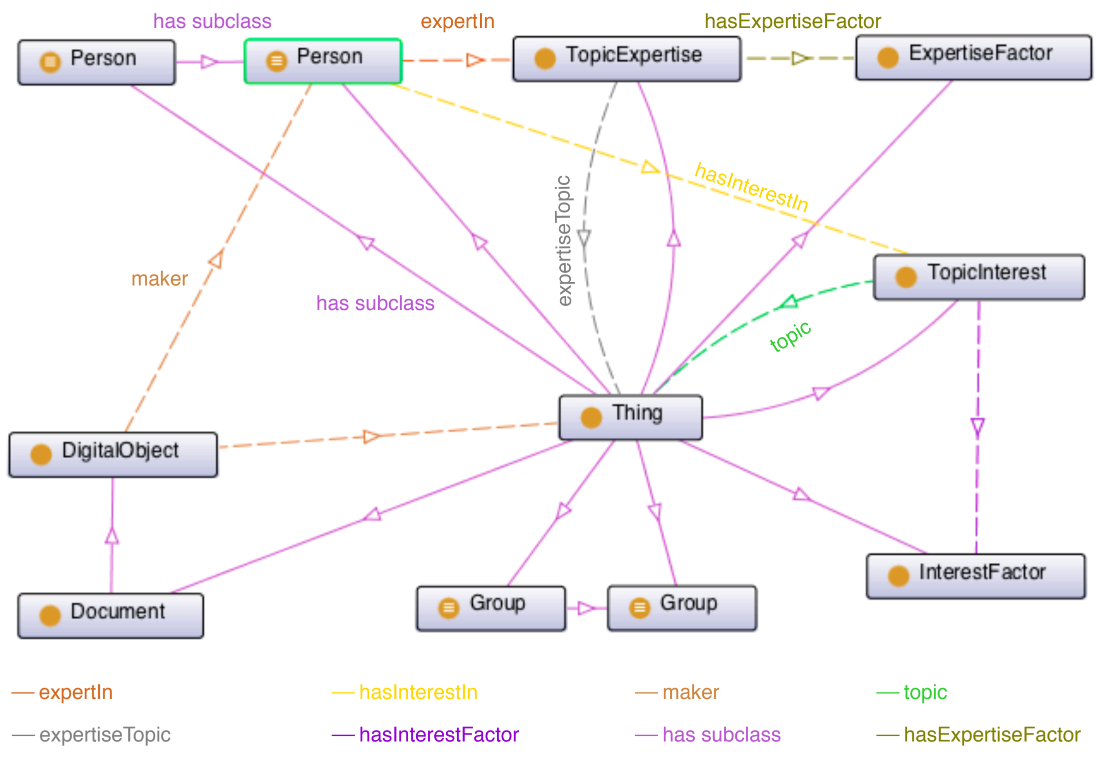


Figure 5.3: Concept Ontology

- `TopicInterest`: represents a persons weighted interest in a particular topic.
- `InterestFactor`: represents the specific interest weight, enabling us to express fuzzy relations.
- `TopicExpertise`: is a weighted expertise for a particular topic.
- `ExpertiseFactor`: represents the specific expertise weight, enabling us to express fuzzy relations.

and relations:

- `maker`: DigitalObject ↔ Person, representing the author/creator of a digital object
- `topic`: DigitalObject ↔ Thing, representing the main topic of a digital object
- `expertIn`: Person ↔ TopicExpertise, representing if a person is an expert for a certain topic

- `hasInterestIn`: Person ↔ TopicInterest, representing if a person is interesting in a certain topic

These relations enable the combination of several semantic representations as shown in Figure 5.2. The `maker` relation signifies that a Person is the author or creator of a Digital Object. This information can be extracted in different ways, depending on the format of the Digital Object, as mentioned in Section 4.2:

- **PDF**: As specified in the PDF reference, a PDF file contains a *document information dictionary* containing the author name as text string [4, p.550], shown in Listing 5.1.

```
1 0 obj
  << /Title (Semantic Space Study)
      /Author (H.L. Truong, J. Schleicher, C. Inzinger, S. Dustdar↩
          )
      /Creator (LaTex with hyperref package)
      /Producer (pdfTeX−1.40.9)
      /CreationDate (D:20100502110347+02'00')
      /ModDate (D:20100503153925+02'00')
  >>
endobj
```

Listing 5.1: Sample PDF document information dictionary

- **Office Open XML** The Office Open XML standard [39] specifies, that document core properties, including author information, are stored in a file `docProps/core.xml` within the Office Open XML document package. It uses the Dublin Core Metadata element set and metadata terms [58] to store document properties in said XML file, as shown in Listing 5.2.

```
<?xml version="1.0" encoding="UTF−8" standalone="yes"?>
<cp:coreProperties
  xmlns:cp="http://schemas.openxmlformats.org/package/2006/↩
      metadata/core−properties"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance">
 <dc:title>Semantic Space Study</dc:title>
 <dc:creator>Truong, Schleicher, Inzinger, Dustdar</dc:creator>
 <dcterms:created xsi:type="dcterms:W3CDTF">2008−06−19T20:00:00Z</↩
     dcterms:created>
 <dcterms:modified xsi:type="dcterms:W3CDTF">2008−06−19T20:42:00Z<↩
     /dcterms:modified>
</cp:coreProperties>
```

Listing 5.2: Sample Office Open XML document properties

- **OASIS Open Document** The OpenDocument specification [57], similar to Open XML above, also defines that document metadata is stored in a separate file, `meta.xml`, and also uses the Dublin Core term vocabulary, as shown in Listing 5.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<office:document-meta
  xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:meta="urn:oasis:names:tc:opendocument:xmlns:meta:1.0"
  xmlns:ooo="http://openoffice.org/2004/office"
  office:version="1.0">
  <office:meta>
    <meta:generator>OpenOffice.org/2.0 [...]</meta:generator>
    <meta:creation-date>2010-05-02T20:10:12</meta:creation-date>
    <dc:creator>Truong, Schleicher, Inzinger, Dustdar</dc:creator>
    <dc:date>2010-05-02T20:14:43</dc:date>
    <dc:language>en-US</dc:language>
  </office:meta>
</office:document-meta>
```

Listing 5.3: Sample OpenDocument metadata

The `topic` relation signifies, that a Digital Object is associated with certain topics, that are elements of an ontology. The principal method of extracting that kind of information is Named Entity Recognition [27; 109; 30; 76; 43; 48; 51; 60], which also is one of the primary topics of the *Conference on Computational Natural Language Learning*[1]. Named Entity Recognition using ontological gazetteers is supported by the GATE toolkit using the *OntoGazetteer* module of the ANNIE component.

As seen above, we also introduce classes to model weighted topic interest and expertise. These concepts can be used to perform additional inference processes, like shared interest discovery [84], expertise modeling [105], and expert finding [35; 41; 14], but will not be primary topics of this thesis.

[1] `http://www.cnts.ua.ac.be/conll/`

5.1.2 *Utilizing the linked data model for different domains*

In order to achieve the desired ability to link disparate semantic representations of digital objects, our data model is based on the RDF graph model.

The RDF model suits our purpose especially well, as the ability to interconnect semantic definitions from disparate sources is an integral part of the specification.

To illustrate the proposed data model, we provide exemplary RDF representation for a digital object showing the integration of two different specialized domains. Figure 5.4 shows an example for the space domain, and Figure 5.5 shows an example for the medical domain.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-↩
    syntax-ns#"
          xmlns:foaf="http://xmlns.com/foaf/0.1"
          xmlns:dc="http://purl.org/dc/elements/1.1/"
          xmlns:phys="http://sweet.jpl.nasa.gov/2.0/phys.↩
              owl"
          xmlns:common_sense="http://www.blackwhale.at/↩
              ontologies/2010/06/common_sense.rdf">

  <rdf:Description rdf:about="http://www.example.org/↩
      document1">
    <dc:title>Report on Reports</dc:title>
    <common_sense:maker rdf:resource="http://www.example.↩
        org/staffid/123"/>
    <dc:description>A textual abstract could go here</↩
        dc:description>
  </rdf:Description>

  <common_sense:Person rdf:ID="http://www.example.org/↩
      staffid/123">
    <foaf:name>John J. Doe</foaf:name>
    <foaf:homepage rdf:resource="http://www.example.org/↩
        staff/jdoe" />
  </common_sense:Person>
  <rdf:Description rdf:about="&phys;PhysicalPhenomena">
      <foaf:primaryTopicOf rdf:resource="http://www.↩
         example.org/document1" />
  </rdf:Description>
</rdf:RDF>
```

Listing 5.4: Sample representation of a digital object, SWEET as domain specific ontology

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-↩
    syntax-ns#"
         xmlns:foaf="http://xmlns.com/foaf/0.1"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:galen="http://krono.act.uji.es/Links/↩
            ontologies/galen.owl"
         xmlns:common_sense="http://www.blackwhale.at/↩
            ontologies/2010/06/common_sense.rdf">

  <rdf:Description rdf:about="http://www.example.org/↩
     document1">
    <dc:title>Report on Medical Things</dc:title>
    <common_sense:maker rdf:resource="http://www.example.↩
       org/staffid/123"/>
    <dc:description>A textual abstract could go here</↩
       dc:description>
  </rdf:Description>

  <common_sense:Person rdf:ID="http://www.example.org/↩
     staffid/123">
    <foaf:name>John J. Doe</foaf:name>
    <foaf:homepage rdf:resource="http://www.example.org/↩
       staff/jdoe" />
  </common_sense:Person>

  <rdf:Description rdf:about="&galen;↩
     AcuteAnteroseptalMyocardialInfarction">
      <foaf:primaryTopicOf rdf:resource="http://www.↩
```

```
            example.org/document1" />
    </rdf:Description>
</rdf:RDF>
```

Listing 5.5: Sample representation of a digital object, GALEN as domain specific ontology

As seen in the examples above it is easy to incorporate any domain specific ontology in the OWL format via our utilization of Thing as a common base class. Further we do not specify any mandatory relationships so that only parts of the model can be used, therefore enabling, for example, omission of social information if not relevant for the specialized domain as shown in Figure 5.6.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-↩
   syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:phys="http://sweet.jpl.nasa.gov/2.0/phys.↩
            owl"
         xmlns:common_sense="http://www.blackwhale.at/↩
            ontologies/2010/06/common_sense.rdf">

    <rdf:Description rdf:about="http://www.example.org/↩
       document1">
      <dc:title>Report on Reports</dc:title>
      <common_sense:maker rdf:resource="http://www.example.↩
         org/staffid/123"/>
      <common_sense:topic rdf:resource="&phys;↩
         PhysicalPhenomena"/>
      <dc:description>A textual abstract could go here</↩
         dc:description>
    </rdf:Description>

</rdf:RDF>
```

Listing 5.6: Sample representation of a digital object, SWEET as domain specific ontology, omitting social information

5.2 Architectural Design of the Concept Search Framework

This section covers the specific conceptual architecture, based on our previous findings. It gives a brief overview, and introduces detailed component and sequence diagrams, as well as different possible implementation scenarios for each component and module. Furthermore, we provide a detailed API description of each component.

5.2.1 *Overview of The Proposed Concept Search Framework*

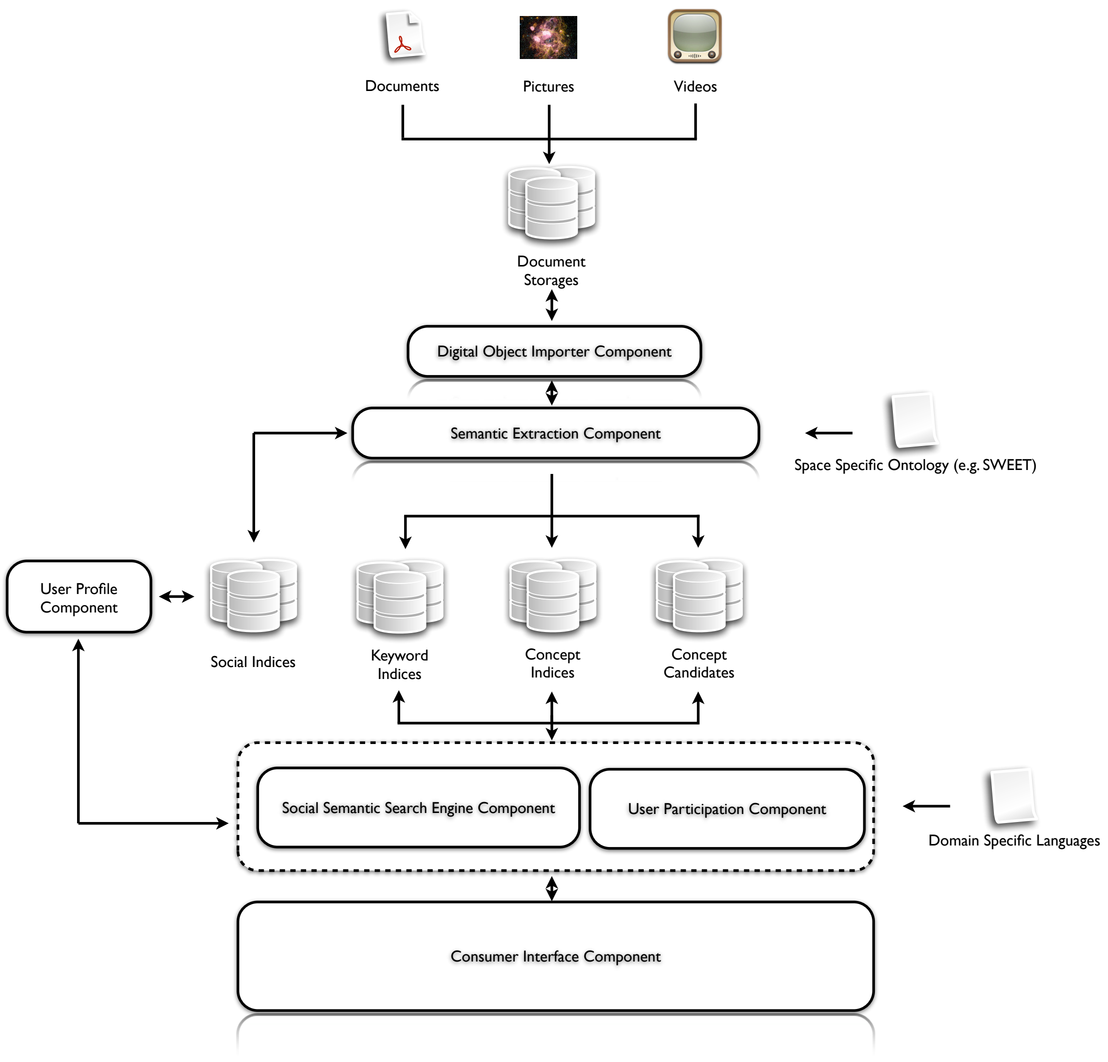

Documents
Pictures
Videos
Document Storages
Digital Object Importer Component
Semantic Extraction Component
Space Specific Ontology (e.g. SWEET)
User Profile Component
Social Indices
Keyword Indices
Concept Indices
Concept Candidates
Social Semantic Search Engine Component
User Participation Component
Domain Specific Languages
Consumer Interface Component

Figure 5.4: Conceptual Architecture Overview

Figure 5.4 depicts the overview of the proposed concept search framework. The framework is based on highly modular component/broker model, and includes the following five core components:

- *Digital Object Importer Component* handles the import of digital objects from various sources into the framework.
- *Semantic Extraction Component* is concerned with extracting semantic information from the imported digital objects
- *Search Engine Component* handles search and result aggregations
- *Consumer Interface Component* is an abstraction layer, providing the general API for the framework as well as certain necessary conversions for later visualization.

One of the main requirements for the concept search framework is that we must provide high flexibility and extensibility in terms of specific component/module implementations, in order to adapt to the specific requirements of the domain as well as to incorporate future developments. To enable this we have to respect a plethora of different execution models. Our design, currently supports different *execution scenarios*, *execution models*, *distribution models*, and utilized *broker modules*.

- *execution scenarios*: meaning whether a component or model is an executable, library or service
- *execution models*: concerned with how an execution is started, triggered or batch processing
- *distribution scenarios*: concerned with how components and modules are distributed

Execution Scenario

In the concept search framework, different execution scenarios exists in terms of the behavior of a module or component

Executable The module itself can be an external executable or script of any kind controlled by a list of parameters or other RPC-methods.

Library The module is written as part of the system, using 3rd party libraries but executed inside the system, and controlled by method calls.

Service The module is an external and autonomous service, accessed via a service API like SOAP, REST or XML-RPC.

Execution Model

Batch Execution Module execution starts at a specified time, or is initiated by a user.

Triggered Execution Modules register for events signifying addition, update, and/or removal. Execution is triggered by these events.

Distribution scenarios

In addition to the different execution scenarios, there are a number of *distribution scenarios* suitable for modules and the according components:

Module: In-Process, Single-Threaded, Centralized Component All modules are executed in the same process context as the component. The component sequentially wakes all modules to be started for batch execution after the designated time, execution can be delayed due to the component having to wait for a previous module to finish. Event based modules can only be incorporated in a single-threaded model if a signal handling method is used to signify that a certain module should be executed.

Module: In-Process, Multi-Threaded, Centralized Component In a multi-threaded component, batch-mode modules can be executed at their designated times (or at least close to their designated times, depending on available system resources), and event-based modules can automatically manage their execution based on external triggers. In this scenario, multiple modules may access the digital object importer API at (almost) the same time, hence concurrency issues have to be taken into account in the component. To manage concurrent access to a single resource, several approaches have been developed, from simple semaphores [34] to ACID transactions [46] and multi-version concurrency control [16].

Module: In-Process, Single-Threaded, Distributed Component Modules are executed like in scenario *Module: In-Process, Single-Threaded, Centralized Component* above, but multiple instances of the component are running concurrently. In this scenario, the concurrently accessed resource is the component accessed next in the framework, and concurrency issues are to be handled in that component. The modules to be executed are distributed over the component instances to be started. Additional instances can be added to a running system.

Module: In-Process, Multi-Threaded, Distributed Component This scenario represents the distributed counterpart to scenario *Module: In-Process, Multi-Threaded, Centralized Component* above, thus combining the multi-threaded scenario with distributed instances of the component. The modules to be executed are distributed over the component instances to be started, and additional instances can be instantiated on-the-fly, should additional modules need to be executed. In this case, concurrency issues have to be managed both at the component API level, as well as at all other accessed components.

Module: Out-of-Process, Single-Threaded, Centralized Component In the context of this description, *out-of-process* denotes a module running completely separated from the component, i.e. the module accesses the component using a service API like XML-RPC [67], REST [42], or SOAP [47]. In this scenario, the component has only limited control over the execution behavior of the modules. To realize batch execution, modules can register a callback handle in the component to be notified when a batch-run is to be performed. Event-based modules are responsible to manage their own execution. The single-threaded, centralized component will sequentially handle requests from the modules, locking out others when a request is being processed. Performance-wise this scenario will be similar to scenario *Module: In-Process, Single-Threaded, Centralized Component*.

Module: Out-of-Process, Multi-Threaded, Centralized Component This scenario represents the combination of scenarios *Module: In-Process, Multi-Threaded, Centralized Component* and *Module: Out-of-Process, Single-Threaded, Centralized Component*. As described above, the component has only limited control over the module's execution behavior, and batch execution can be handled by modules registering callback handles in the component. The multi-threaded centralized component allows for multiple modules to access the importer API concurrently.

Module: Out-of-Process, Single-Threaded, Distributed Component The component in this scenario is distributed as in scenario *Module: In-Process, Single-Threaded, Distributed Component*, i.e. multiple instances, capable of serving one request at a time via the service API. The modules can contact any instance of the component to fulfill their request, and can also try different instances to facilitate load balancing or failover. However, as the component has effectively no control over which module accesses which component-instance, it can only provide recommendations on which modules access which instance to balance the load as evenly as possible. These recommendations can of course only be generated if the component instances are aware of each other, which is not mandatory for the basic functionality of the scenario, but an optimizing extension.

Module: Out-of-Process, Multi-Threaded, Distributed Component This scenario is similar to *Module: In-Process, Multi-Threaded, Distributed Component*, in that multiple instances of the component are instantiated, capable of serving multiple concurrent requests. As in scenario *Module: Out-of-Process, Single-Threaded, Distributed Component*, the modules can contact any instance of the component to serve their requests, and the instances can give recommendations on which instances best accessed in subsequent requests.

Module: In-Process & Out-of-Process In addition to the scenarios outlined above, the component can also allow for a mixed model of in-process and out-of-process modules.

Broker Module

The *Broker* module is responsible for the coordination and execution of modules, as well as holding and handing off received information to appropriate sibling components. To facilitate the coordination of modules, the *Broker* also acts as a module registry, where modules signal their availability for service, and announce their functional and configuration capabilities.

Brokers of the different components will have different additional APIs, and are described in detail with the respective component.

5.2.2 *Digital Object Importer Component*

The first component of consideration is the digital object importer (DOIC). Its role is to import digital objects from different digital object stores for further processing. The key design goal is to provide a defined and canonical abstraction of the gathered information for the system, so that components using the imported digital objects (i.e. the Semantic Extraction Component) need not be concerned about how to obtain digital objects from their digital object stores (e.g. *file system*, *web site*, *database*).

Another design goal of this component is to allow easy extensibility, which is realized by allowing the addition of new importer modules to support additional data stores and digital object formats. By proposing this paradigm we ensure to meet the requirements stated in Supported Digital Objects.

Component Architecture

The DOIC allows for executing *Digital Object Importer Modules* (DOIM), in order to provide unified means for processing the available information, by converging different *input formats* and *channels* into a canonical representation of the digital objects.

Figure 5.5 shows an overview of the component and its interaction with the DOIM. As mentioned above, the DOIC is responsible for handling the transition of digital objects from an external data store (outlined as Document Storage in the diagram) into the system, represented in a canonical format, ready for processing.

The component provides the DOIMs with a simple and easy to use *Digital Object Container API* (see Section 5.2.2) for importing digital objects, as well as storing state information for already imported documents, in order to minimize successive imports of the same object. Furthermore, the DOIC implements methods to manage the registering and discovery of modules. A more detailed overview over the component's API can be found in Section 5.2.2.

— ❧ —

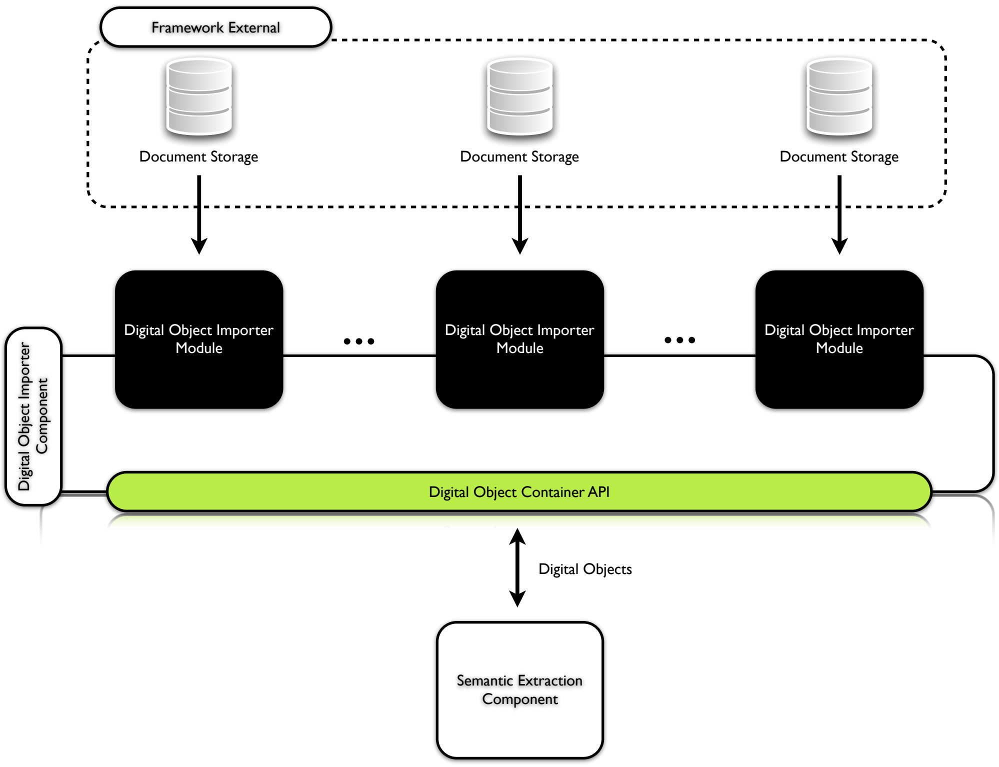

Framework External
Document Storage
Document Storage
Document Storage
Digital Object Importer Module
Digital Object Importer Module
Digital Object Importer Module
Digital Object Importer Component
Digital Object Container API
Digital Objects
Semantic Extraction Component

Figure 5.5: Component diagram of the Digital Object Importer

The following paragraphs outline the structure of a *Digital Object Importer Module* in greater detail:

Digital Object Importer Module A DOIM is responsible for retrieving digital objects from their data store, in order to make them processable by the system. The component diagram in Figure 5.6 shows an exemplary Web crawler module. The digital object importer component's Broker module provides a digital object state store (via the *Digital Object Container API*), enabling importer modules to maintain information about the state of already processed digital objects inside the system. The imported digital objects are handed off to the Semantic Extraction Component.

- `Input`: As the digital object importer modules provide the entry-point for information into the system, only few requirements exist for the input side of a module: Support for importing one or more of the formats identified in the requirements in Section 4.3.1 from a digital object store such as *file system*, *web site*, or *database*, and an optional configuration interface to modify the module's behavior.

- `Execution Model`: Batch execution, as well as triggered execution are suitable for importer modules. When a batch execution model is used, it is important that the module does not unnecessarily import a digital object multiple times. The DOIC provides a digital object state store for modules, so that multiple imports of the same (unchanged) digital object can be prevented. If the module is executed outside of the scope of the digital object importer component, it can only use the digital object state store, if the component exposes its interface via a service API.

- `Output`: An importer module should either provide the system with a serialized version of the digital object, or provide a system-accessible URI.

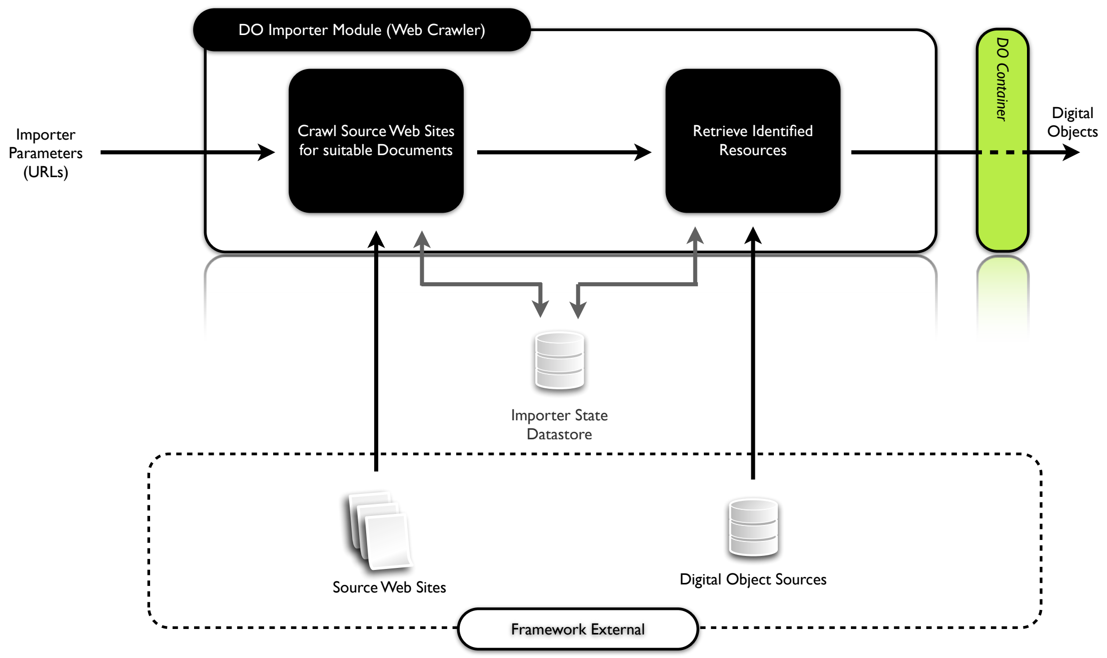

DO Importer Module (Web Crawler)
Importer Parameters (URLs)
Crawl Source Web Sites for suitable Documents
Retrieve Identified Resources
DO Container
Digital Objects
Importer State Datastore
Source Web Sites
Digital Object Sources
Framework External

Figure 5.6: Component diagram of a Digital Object Importer Module

A more detailed description of the interface required for DOIMs can be found in the following section.

Interface Description

In the context of the DOIC we have to examine two interfaces, the *Digital Object Container API*, and the *Document Importer Module Support API*.

Digital Object Container API This interface represents the primary API of the DOIC, and provides methods for registering and deregistering modules, as well as adding, updating and deleting digital objects to/from the system. Detailed descriptions of the methods can be found in Figures 5.7 and 5.8.

Operation name	**Description and comments**
Module Lifecycle Management	
Register Module	This operation is used to register an importer module with the component for later use. By registering, a module signals availability for service.
Deregister Module	This operation allows a module to disconnect from the component, and gracefully stop its operations.
Digital Object Manipulation	
Add Digital Object	This operation is used by importer modules to add a new digital object to the component's digital object container.
Update Digital Object	This operation allows a module to mark an existing digital object as updated.
Delete Digital Object	This operation is used by importer modules to mark a digital object for deletion.

Figure 5.7: Document Importer Component Interface overview

Importer Module Support API This interface allows importer modules to store arbitrary state information about digital objects, to prevent them from unnecessarily importing objects multiple times. Detailed descriptions can be found in Figures 5.9 and 5.10.

Parameter name	Type	Direction	Description and comments
Authorization Key	`String`	IN	The module's authorization key
Module ID	`Integer`	OUT	The handle to the module if it was registered successfully, `null` otherwise

(a) Operation `Register Module`

Parameter name	Type	Direction	Description and comments
Authorization Key	`String`	IN	The module's authorization key
Module ID	`Integer`	IN	The handle to the module as obtained by a call to `registerModule`
Status	`Boolean`	OUT	`true` if module was deregistered successfully, `false` otherwise

(b) Operation `Deregister Module`

Parameter name	Type	Direction	Description and comments
Digital Object	`String <XML>`	IN	The serialized representation of the digital object or a system-accessible link to the digital object to be added
Digital Obj. GUID	`String`	IN	The handle to the digital object within the system

(c) Operation `Add Digital Object`

Parameter name	Type	Direction	Description and comments
Digital Obj. GUID	`String`	IN	The handle to the digital object to be marked as updated
Digital Object	`String <XML>`	IN	The serialized representation of the digital object or a system-accessible link to the updated digital object
Status	`Boolean`	OUT	`true` if updated successfully, `false` otherwise

(d) Operation `Update Digital Object`

Figure 5.8: Document Importer Component Interface details

Parameter name	**Type**	**Direction**	**Description and comments**
Digital Obj. GUID	`String`	IN	The handle to the digital object to be marked for deletion
Status	`Boolean`	OUT	`true` if digital object was deleted successfully, `false` otherwise

(e) Operation `Delete Digital Object`

Figure 5.8: Document Importer Component Interface details (contd.)

Operation name	**Description and comments**
Set Digital Object State	This operation allows an importer module to store arbitrary state information for the specified digital object, in order to prevent unnecessary imports of the object.
Get Digital Object State	This operation allows an importer module to retrieve any stored state information about the specified digital object.

Figure 5.9: Document Importer Module Support Interface overview

Parameter name	**Type**	**Direction**	**Description and comments**
Digital Obj. GUID	`String`	IN	The handle to the digital object to set the state information for
State Information	`String<XML>`	IN	Arbitrary state information in XML-serialized form
Status	`Boolean`	OUT	`true` if state saved successfully, `false` otherwise

(a) Operation `Set Digital Object State`

Parameter name	**Type**	**Direction**	**Description and comments**
Digital Obj. GUID	`String`	IN	The handle to the digital object to retrieve the state information for
State Information	`String<XML>`	OUT	Arbitrary state information in XML-serialized form
Status	`Boolean`	OUT	`true` if state retrieved successfully, `false` otherwise

(b) Operation `Get Digital Object State`

Figure 5.10: Document Importer Module Support Interface details

Sequence Diagram

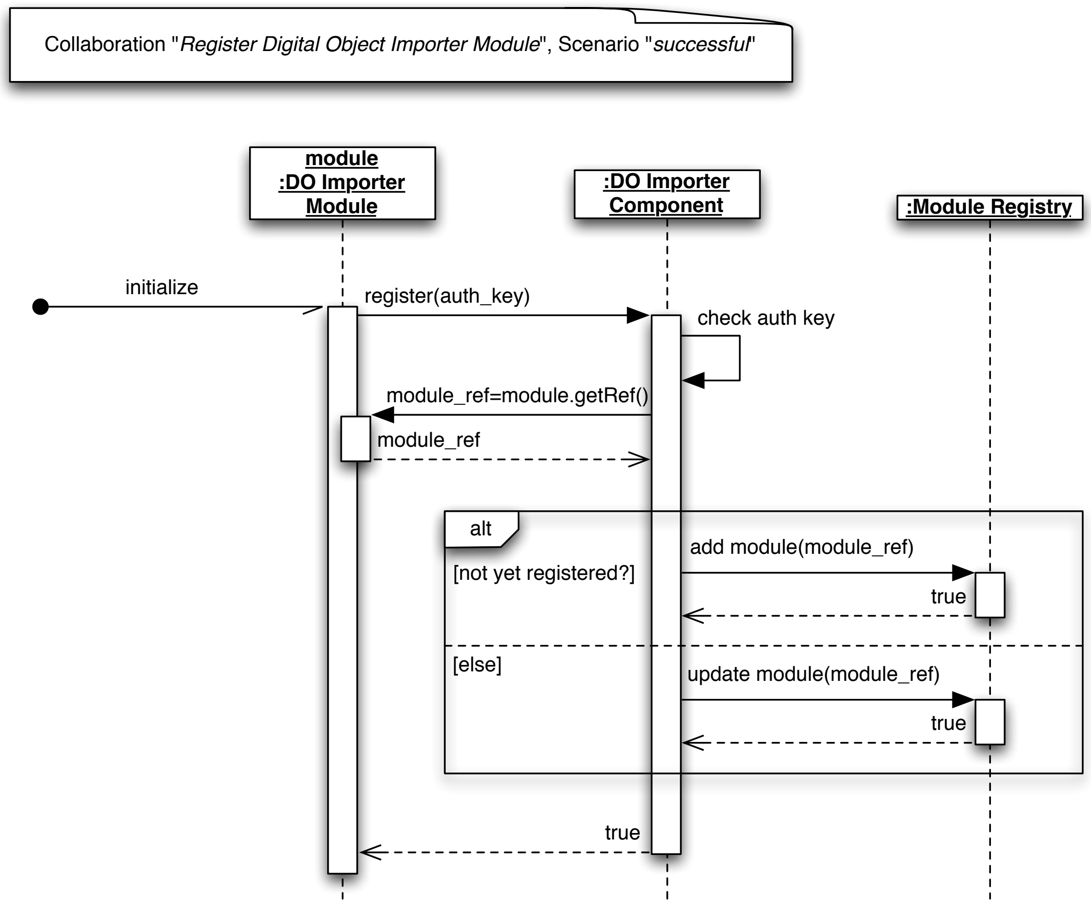

Collaboration "*Register Digital Object Importer Module*", Scenario "*successful*"
module :DO Importer Module
:DO Importer Component
:Module Registry
initialize
register(auth_key)
check auth key
module_ref=module.getRef()
module_ref
alt
[not yet registered?]
add module(module_ref)
true
[else]
update module(module_ref)
true
true

Figure 5.11: Sequence diagram showing a DO importer module successfully registering with the component

Figure 5.11 shows an exemplary Sequence Diagram for a successful Digital Object Importer Module registration.

5.2.3 *Semantic Extraction Component*

The Semantic Extraction Component (sxc) is responsible for processing the digital objects imported by the Digital Object Importer Component, in order to generate syntactic and semantic annotations for later retrieval, and is also implemented as a Broker model. The component furthermore ensures that the generated annotations are stored in an efficient manner, so that successive retrieval processes can be completed in a performant manner.

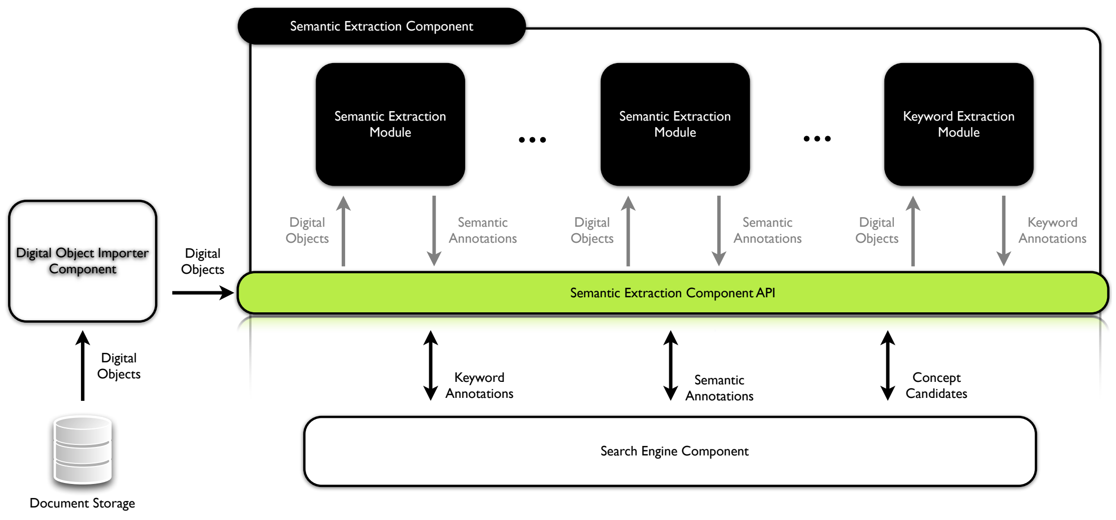


Figure 5.12: Component Diagram of the Semantic Extraction Component

Component Architecture

The *Semantic Extraction Component* processes the digital objects imported by the *Digital Object Importer Component*, and feeds them to all registered *semantic extraction modules* (SXM) capable of handling the specific digital object type. It allows the execution of *semantic extraction modules* to provide syntactic and semantic annotations for a digital object, and hand them off to the *Search Engine Component* via the Broker's API. The component diagram in Figure 5.12 shows the interactions of the component with its modules. The component provides SXMs with an API for storing keyword annotations, concept annotations, and concept candidates extracted from processed digital objects. A more detailed description of the component's APIs can be found in Section 5.2.3.

— ❧ —

The following paragraphs outline the structure of a *Semantic Extraction Module* in greater detail:

Semantic Extraction Module A *Semantic Extraction Module* is responsible for extracting keyword annotations, concept annotations, or concept candidates from imported digital objects, in order to feed the system's indices for later retrieval operations. The component diagrams in Figure 5.13 show exemplary extraction modules.

In order to satisfy the requirement of a combined semantic representation (see Figure 5.2 and Supported Extraction Mechanisms), which is necessary to support specialized domains, the SXM is responsible for extracting the necessary information, in order to fill the data model. Different SXMs can extract different parts of the model. Possible approaches to do so have been shown in Section 5.1.1.

- `Input`: Depending on the execution model, the module either receives a single digital object for processing, or a collection thereof. The digital object can be represented in a serialized form, or as a canonical, system-accessible URI ready for processing. If the module is not able to retrieve the digital object by URI, it can instruct the Broker to do so by setting the according option when registering with the component.
- `Execution Model`: Batch execution, as well as triggered execution are suitable for SXMs. Furthermore, as the extraction process is largely self-contained, the modules can be executed in-process or out-of-process.
- `Output`: After processing a digital object, the semantic extraction module can utilize the Broker's API to create annotations related to the digital object. It further ensures adherence to the necessary standard format for the corresponding annotation class (keyword, semantic, social), to meet the requirements specified in 4.3. Another approach could use a previously agreed-upon XML annotation format, returned by the extraction module as result of its processing, which the broker then processes and stores in the according indices via the *Search Engine Component*.

A more detailed description of the interface required for SXMs can be found in the following section.

Interface Description

In the context of the SXC we will examine two interfaces, the *component API*, as well as the *Semantic Extraction Module API*.

Component API This interface represents the primary API of the SXC, and provides methods for registering and deregistering modules, as well as storing keyword annotations, concept annotations, and concept candidates. Detailed descriptions can be found in Figures 5.14 and 5.15.

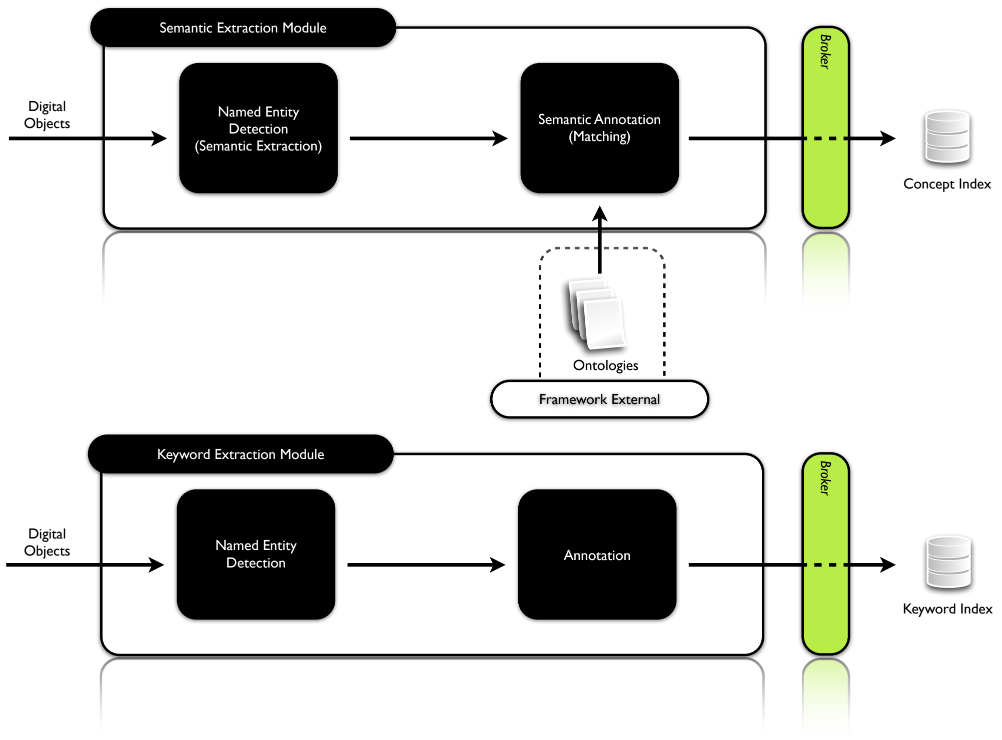


Figure 5.13: Exemplary Semantic Extraction Modules

Module API This interface describes the operations, that must be supported by a *Semantic Extraction Module*. Detailed descriptions can be found in Figures 5.16 and 5.17.

Sequence Diagram

Figure 5.18 shows an exemplary Sequence diagram for the semantic annotation of a DO using named entity detection and semantic matching.

Operation name	**Description and comments**
Module Lifecycle Management	
Register Module	This operation is used to register an extraction module with the component for later use. By registering, a module signals availability for service.
Deregister Module	This operation allows a module to disconnect from the component, and gracefully stop its operations.
Annotation Storage	
Store Keyword Annotation	The operation allows a module to store a keyword annotation in the system's keyword indices.
Store Concept Annotation	The operation allows a module to store a concept annotation in the system's concept indices.
Store Concept Candidate	The operation allows a module to store a concept candidate in the system's concept candidate store.

Figure 5.14: Semantic Extraction Component Interface overview

Parameter name	Type	Direction	Description and comments
Authorization Key	`String`	IN	The module's authorization key
Module ID	`Integer`	OUT	The handle to the module if it was registered successfully, `null` otherwise

(a) Operation `Register Module`

Parameter name	Type	Direction	Description and comments
Authorization Key	`String`	IN	The module's authorization key
Module ID	`Integer`	IN	The handle to the module as obtained by a call to `registerModule`
Status	`Boolean`	OUT	`true` if module was deregistered successfully, `false` otherwise

(b) Operation `Deregister Module`

Parameter name	Type	Direction	Description and comments
Keyword Annotation	`String <XML>`	IN	The keyword annotation to store in an XML-serialized form.
Status	`Boolean`	OUT	`true` if annotation was stored successfully, `false` otherwise

(c) Operation `Store Keyword Annotation`

Parameter name	Type	Direction	Description and comments
Concept Annotation	`RDF-XML`	IN	The concept annotation to store in an XML-serialized form.
Status	`Boolean`	OUT	`true` if annotation was stored successfully, `false` otherwise

(d) Operation `Store Concept Annotation`

Figure 5.15: Semantic Extraction Component Interface details

Parameter name	Type	Direction	Description and comments
Concept Candidate	`RDF-XML`	IN	The concept candidate to be stored in XML-serialized form
Status	`Boolean`	OUT	`true` if concept candidate was stored successfully, `false` otherwise

(e) Operation `Store Concept Candidate`

Figure 5.15: Semantic Extraction Component Interface details (contd.)

Operation name	Description and comments
Process Digital Object	This operation is invoked to queue a digital object in the Semantic Extraction Module for processing

Figure 5.16: Semantic Extraction Module Interface overview

Parameter name	Type	Direction	Description and comments
Digital Object	`String <XML>`	IN	The serialized representation of the digital object to be processed
Status	`Boolean`	OUT	`true` if the object was queued successfully, `false` otherwise

(a) Operation `Process Digital Object`

Figure 5.17: Semantic Extraction Module Interface details

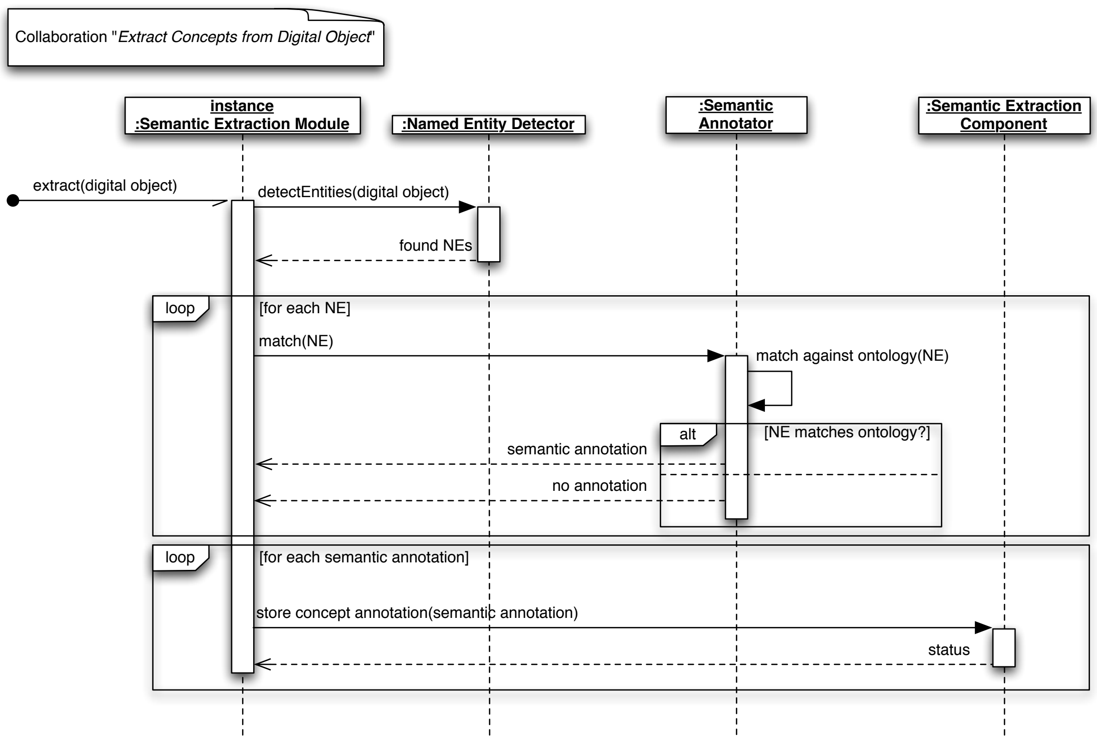

Collaboration "Extract Concepts from Digital Object"
instance :Semantic Extraction Module
:Named Entity Detector
:Semantic Annotator
:Semantic Extraction Component
extract(digital object)
detectEntities(digital object)
found NEs
loop
[for each NE]
match(NE)
match against ontology(NE)
alt
[NE matches ontology?]
semantic annotation
no annotation
loop
[for each semantic annotation]
store concept annotation(semantic annotation)
status

Figure 5.18: Sequence diagram showing semantic extraction of concepts from a DO by a Semantic Extraction Module

5.2.4 *Search Engine Component*

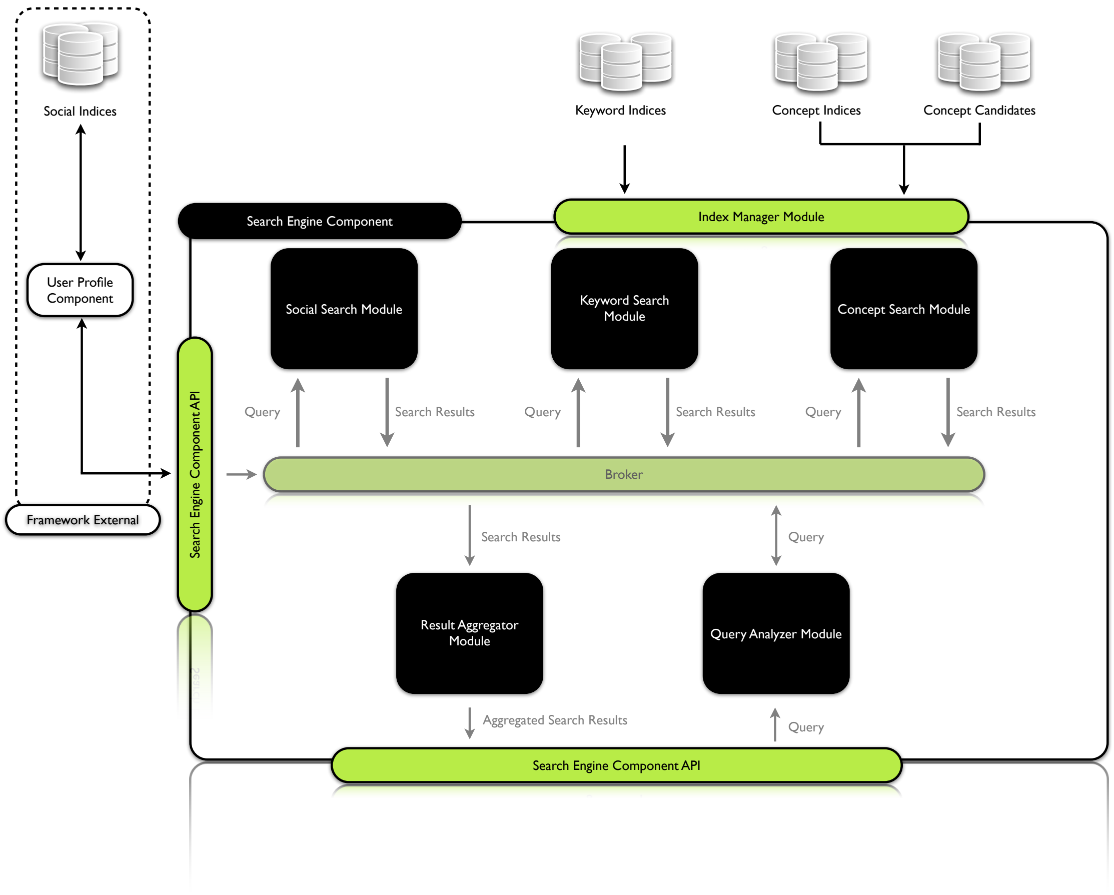


Figure 5.19: Search Engine Component Overview

The search engine component is responsible for executing the search based on a query, and is shown in Figure 5.19. A query is submitted into the *Query Analyzer Module* via the *Search Engine Component API* (Section 5.24), which determines the query type in accordance with registered *Search modules* from the *Broker*. After that, the marked up query is transmitted to the *Broker*, which then submits it to the according *Search module*. Search modules registered with the *Broker* perform the search and return the results to the *Broker*, which in turn submits the results to the *Result Aggregator Module*, which combines the results from different search modules and returns them. Furthermore, the Search Engine Component is concerned with efficiently managing and distributing the system's indices via the *Index Manager Module*.

Despite the `Execution Models` mentioned above for each component, the outlined *Distribution Models* mentioned in Section 5.2.1 apply for the component and its modules, and play an important role for the Search Engine component in terms of scalability and reliability.

Component Architecture

In the following paragraphs we will describe the principal modules composing the component in greater detail.

Broker The search engine component's Broker module provides APIs to manage *Search modules*, *Query analyzers* and *Result aggregators*, as well as to manage indices and its capabilities are outlined in detail in the API description of the component.

Query Analyzer Module The *Query Analyzer module* receives a query via the *Broker* and analyzes it according to the registered *Search modules* in the *Search Engine Component*. There are multiple possible input types for queries.

- `Input`: The *Query Analyzer Module* accepts the following input types:
  - *Keyword Query*: A natural language keyword query, submitted as plain text.
  - *Semantic Query*: A set of concepts submitted as key/value pairs
  - *Social Query*: A social query submitted as key/value pairs.

  Despite the basic three outlined classes for input types, queries can generally be joined via a basic set of operators and regular expressions. A query can also be partly Keyword, Semantic and Social

- `Execution Model`: The only reasonable execution model for the Query Analyzer Module QAM is a triggered execution.

- `Output`: The output of a *Query Analyzer module* is one or more normalized queries, annotated with the type of the query. The type of a query can be *keyword*, *semantic* or *social*, provided as a key/value pair.

Keyword Search Module A Keyword Search Module shown in Figure 5.20 receives a query, annotated as keyword query, from the Query Analyzer Module through the broker. It then runs its internal Query Analyzer on the submitted query, optimizes it accordingly, and performs syntactic matching against a Keyword index via the Index Manager.

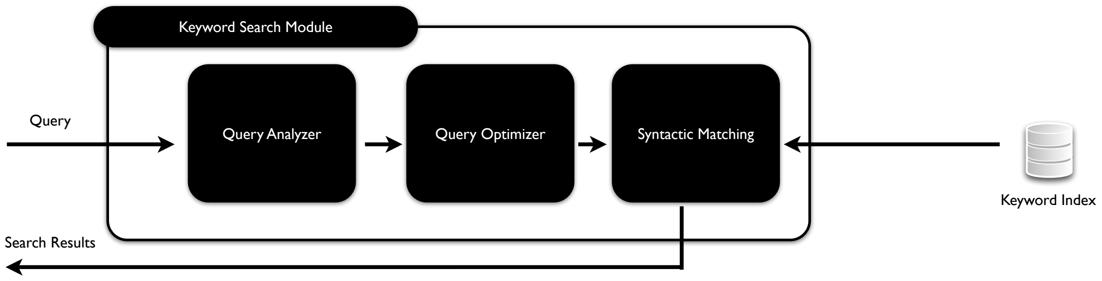


Figure 5.20: Keyword Search Module

- `Input`: The input of a Keyword Search Module is a query in form of a string.
- `Execution Model`: As with every module participating in user triggered interaction the only reasonable execution model is triggered execution.
- `Output`: The output of a Keyword Search Module is a ordered result set. An element of a result set is a key/value pair, where the value is a link to the retrieved document.

Concept Search Module A *Concept Search Module* shown in Figure 5.21, receives a query, annotated as semantic query, from the Query Analyzer Module through the broker. It then runs its internal Query Analyzer on the submitted query, optimizes it accordingly, and performs semantic matching against a Concept index via the Index Manager. Previous research on concept search discusses semantic queries to be submitted via natural language interfaces [59; 79; 90], ontology driven search [17; 92; 87], and ranking [8].

- `Input`: The input of a Concept Search Module is a either a concept as string, or multiple concepts as strings, separated by an agreed separator. Furthermore, SPARQL queries can also be an acceptable input for the Concept Search Module, submitted as string.

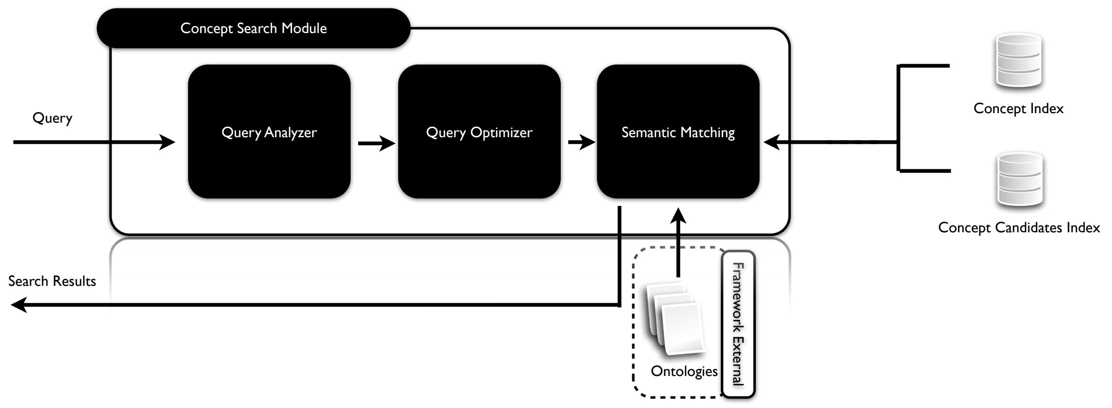


Figure 5.21: Concept Search Module

- `Execution Model`: Also only triggered execution makes sense.
- `Output`: The output of a Concept Search Module is an ordered result set. An element of a result set is a key/value pair, where the value is a link to the retrieved document.

Social Search Module A social Search Module shown in Figure 5.22, receives a query, annotated as social query, from the Query Analyzer Module via the broker. It then runs its internal Query Analyzer on the submitted query, optimizes it accordingly, and performs social matching against a Social index via the Index Manager.

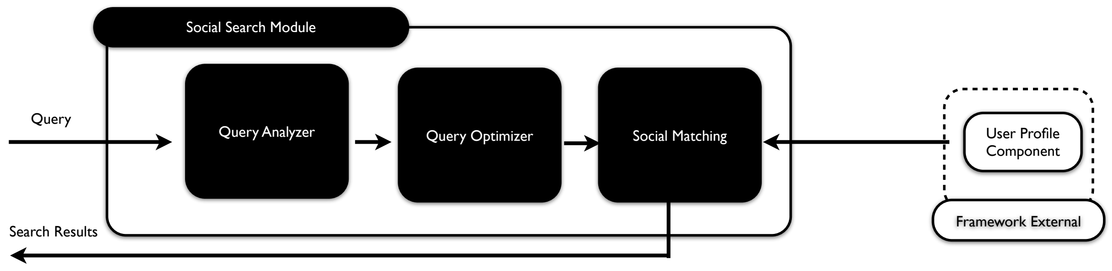


Figure 5.22: Social Search Module

- `Input`: The input of a Social Search Module is either a name or some attribute of a Person, as identified by our Linked Model for Representing Information in the Concept Search Framework submitted as string.
- `Execution Model`: As with all the other Search modules, the only reasonable execution model is a triggered execution.

- `Output`: The output of a Social Search Module is an ordered result set. An element of a result set is a key/value pair, where the value is a link to the retrieved document.

Result Aggregator Module The main role of the Result Aggregator is the ranking and combination of the results of each queried Search Module. Existing research includes [37], [77] and [25].

- `Input`: The Result Aggregator gets multiple result sets consisting of previously described key value pairs, as well as an internal ordering of the result set based on the according query.
- `Execution Model`: Since the Result Aggregation Module is the last step of a user triggered control flow in our architecture the only reasonable execution model is triggered execution.
- `Output`: The output of the Result Aggregator is a ranked/ordered result set.

Index Manager Module The Index Manager Module is part of the Broker, and responsible for managing the indices. Moreover, it is concerned with efficient distribution of indices, as well as storage concerns for concepts and concept indices. In terms of storage and related indices, two different models apply. Keyword indices, which can be handled like [75] by tools such as Apache Lucene and Lemur, and concept as well as concept candidate indices, which are handled via RDF Stores [7], using tools such as AllegroGraph or Sesame. Distribution of indices can either be handled via replication or sharding, as shown in [31].

- `Input`: The Index Manager receives Queries form the Search Modules, as well as Annotations from the Semantic Extraction Component.
- `Execution Model`: For the Index Manager, both triggered and batch execution are feasible for different tasks. Annotation storage can be aggregated and processed in batches for optimization purposes. Retrieval actions are best processed in a triggered manner.
- `Output`: The output of an Index Manager are result sets for a query by a Search Module.

User Profile Component and Domain Specific Languages The *User Profile Component* (UPC) is considered an external resource from the framework and is responsible for maintaining user and social data. Possible examples would be LDAP[23], or a social network via OpenSocial[2]. In addition, the UPC is responsible for storing and aggregating statistical information about the user, as well as its interaction behavior with the framework, utilizing approaches like [54]. Inside the framework, social data is considered similar to concepts. as they are practically identical.

The support of *Domain Specific Languages* DSL is on one hand ensured via the Frameworks API, furthermore, external DSLS can be easily incorporated via framework-external mappers, and therefore need no further elaboration in this report.

Interface Description

In the context of the SEC we have to examine multiple interfaces, the *Digital Object Container API*, and the *Document Importer Module Support API*.

Broker API This interface represents the primary API of the SEC and provides methods for registering and deregistering modules to/from the system. Detailed descriptions of the methods can be found in Figures 5.23 through 5.34.

Module API These interface describe the operations, that must be supported by *Query Analyzer Module*, *Keyword Search Module*, *Semantic Search Module*, *Social Search Module* as well as the *Result Aggregator Module*. Detailed descriptions can be found in Figures 5.25 and 5.26.

[2]http://code.google.com/apis/opensocial/

Operation name	**Description and comments**
Module Lifecycle Management	
Register Module	This operation is used to register an importer module with the component for later use. By registering, a module signals availability for service.
Deregister Module	This operation allows a module to disconnect from the component, and gracefully stop its operations.
Search	
Search	This operation is used to submit a query to the Search Engine Component and start the search process.
Annotation Storage	
Store Keyword Annotation	The operation allows a module to store a keyword annotation in the system's keyword indices.
Store Concept Annotation	The operation allows a module to store a concept annotation in the system's concept indices.
Store Concept Candidate	The operation allows a module to store a concept candidate in the system's concept candidate store.

Figure 5.23: Search Engine Component Interface overview

Parameter name	Type	Direction	Description and comments
Authorization Key	`String`	IN	The module's authorization key
Module Type	`<String, String>`	IN	A key value pair consisting of the type of module being registered, allowed values are `<search,concept>` for a Concept Search Module, `<search,keyword>` for a Keyword Search Module, `<search,social>` for a Social Search Module, as well as `<query,null>` for a Query Analyzer and `<aggregator,null>` for a Result Aggregator.
Module ID	`Integer`	OUT	The handle to the module if it was registered successfully, `null` otherwise

(a) Operation `Register Module`

Parameter name	Type	Direction	Description and comments
Authorization Key	`String`	IN	The module's authorization key
Module ID	`Integer`	IN	The handle to the module as obtained by a call to `registerModule`
Status	`Boolean`	OUT	`true` if module was deregistered successfully, `false` otherwise

(b) Operation `Deregister Module`

Parameter name	Type	Direction	Description and comments
Query	`String`	IN	The combined query in an xml serialized form

(c) Operation `Search`

Figure 5.24: Search Engine Component Interface details

Parameter name	Type	Direction	Description and comments
Keyword Annotation	`String <XML>`	IN	The keyword annotation to store in an XML-serialized form.
Status	`Boolean`	OUT	`true` if annotation was stored successfully, `false` otherwise

(d) Operation `Store Keyword Annotation`

Parameter name	Type	Direction	Description and comments
Concept Annotation	`String <XML>`	IN	The concept annotation to store in an XML-serialized form.
Status	`Boolean`	OUT	`true` if annotation was stored successfully, `false` otherwise

(e) Operation `Store Concept Annotation`

Parameter name	Type	Direction	Description and comments
Concept Candidate	`String<XML>`	IN	The concept candidate to be stored in XML-serialized form
Status	`Boolean`	OUT	`true` if concept candidate was stored successfully, `false` otherwise

(f) Operation `Store Concept Candidate`

Figure 5.24: Search Engine Component Interface details (contd.)

Operation name	Description and comments
Analyze Query	This operation is invoked to analyze an incoming query

Figure 5.25: Query Analyzer Module Interface overview

Parameter name	Type	Direction	Description and comments
Query	`String`	IN	The combined query to be analyzed and split up and/or annotated
Annotated Queries	`[<String, String>]`	OUT	The annotated queries in an array split into key/value pairs according to query type.

(a) Operation `Analyze Query`

Figure 5.26: Query Module Interface details

Operation name	Description and comments
Process Query	This operation is invoked to process an incoming keyword query

Figure 5.27: Keyword Search Module Interface overview

Parameter name	Type	Direction	Description and comments
Query	`<String, String>`	IN	A query annotated as Keyword Query in the form of `<keyword,query>`
Result set	`[<String, String>]`	OUT	The result set for the query in an array split into key/value pairs according to query type.

(a) Operation `Process Query`

Figure 5.28: Keyword Search Interface details

Operation name	Description and comments
Process Query	This operation is invoked to process an incoming keyword query

Figure 5.29: Concept Search Module Interface overview

Parameter name	Type	Direction	Description and comments
Query	`<String, String>`	IN	A query annotated as Concept Query in the form of `<concept,query>`
Result set	`[<String, String>]`	OUT	The result set for the query in an array split into key/value pairs according to query type.

(a) Operation `Process Query`

Figure 5.30: Concept Search Interface details

Operation name	Description and comments
Process Query	This operation is invoked to process an incoming keyword query

Figure 5.31: Social Search Module Interface overview

Parameter name	Type	Direction	Description and comments
Query	`<String, String>`	IN	A query annotated as Social Query in the form of `<social,query>`
Result set	`[<String, String>]`	OUT	The result set for the query in an array split into key/value pairs according to query type.

(a) Operation `Process Query`

Figure 5.32: Social Search Interface details

Operation name	Description and comments
Aggregate Results	This operation is invoked to combine results from different Search Modules

Figure 5.33: Result Aggregator Module Interface overview

Parameter name	**Type**	**Direction**	**Description and comments**
Results	`[<String, String>]`	IN	An array of results, where each result is annotated as either `<keyword,result>`, `<concept,result>` or `<social,result>`.
Result set	`[<String, String>]`	OUT	The result set for the initial query in an array split into key/value pairs according to query type, ranked and aggregated.

(a) Operation `Aggregate Results`

Figure 5.34: Result Aggregator Interface details

Sequence Diagram

Figure 5.35 shows the sequence diagram for the Search Engine Component. Specifically, it presents a successful search by a combined keyword and concept query, querying the respective indices and retrieving the found results.

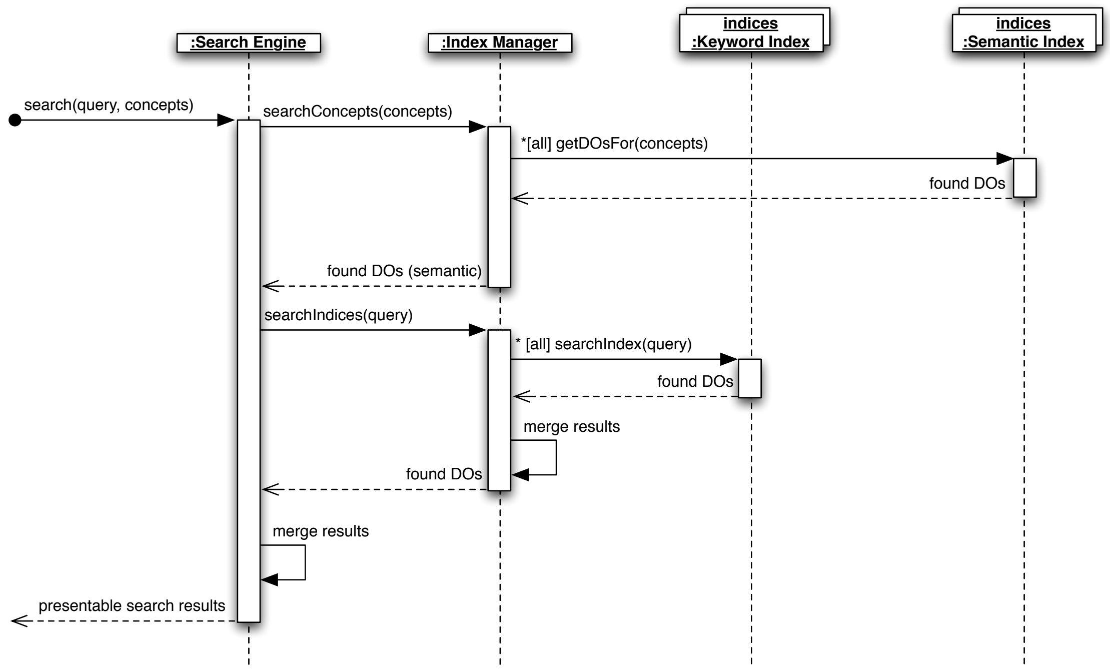


Figure 5.35: Sequence Diagram for a successful search operation using a combined query via the Search Engine Component

5.2.5 User Participation Component

The *User Participation Component* is responsible for handling all elements of user participation in our Semantic search framework. Figure 5.36 shows an overview of the component and its modules.

The main role of the component is to provide means for adapting and adding semantic and social representations [53], as well as to enable feedback mechanisms like ranking, and the integration of alternative computation models like human computation, or game based approaches, as shown in the works of [98].

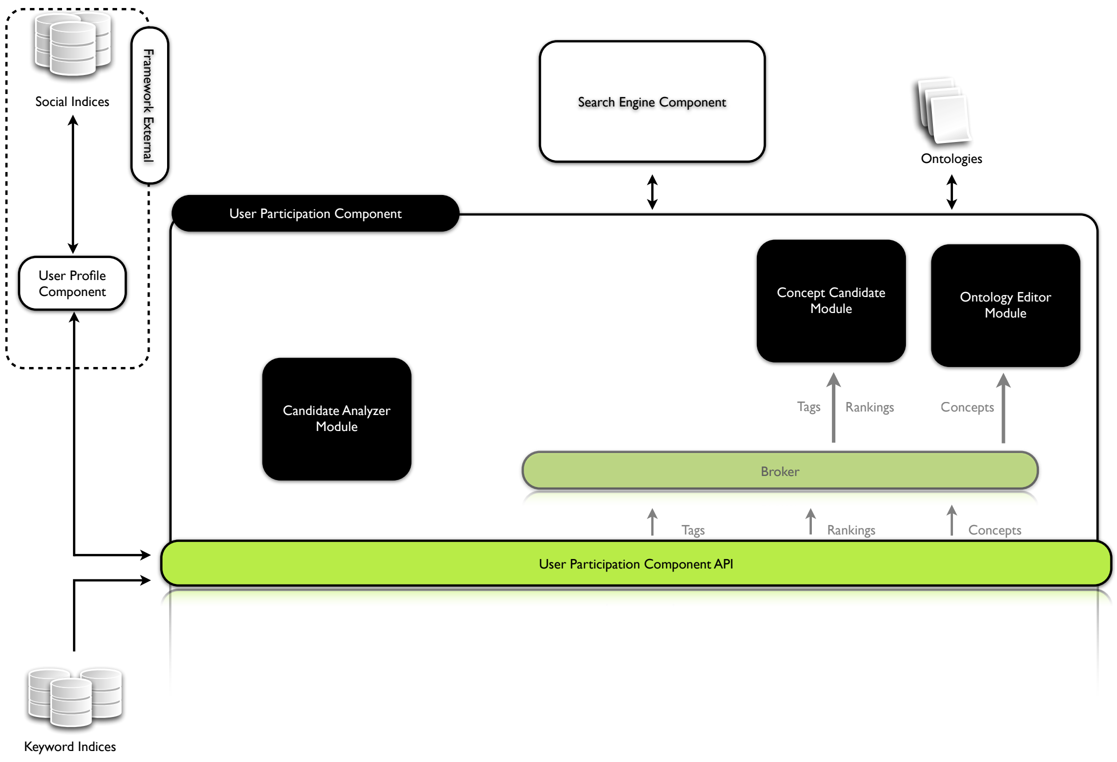


Figure 5.36: User Participation Component Overview

Component Architecture

In the following paragraphs we will describe the principal modules composing the component in greater detail.

Candidate Analyzer Module The *Candidate Analyzer* is responsible for analyzing keywords from the Keyword indices, and mark them as possible candidates to modify or augment existing semantic representations. It does so by utilizing statistical analysis over occurrence and usage patterns of keywords, in order to mark them as possible concept candidates. Examples can be found in [91].

- `Input`: The input of a *Candidate Analyzer Module*, are keywords from the Keyword index, provided in a formerly agreed upon XML representations that contains, the keywords as well as statistical information like occurrence usage patterns about those keywords.
- `Execution Model`: The *Candidate Analyzer Module* can be run as batch execution in certain time intervals, or it can be triggered by a user action.

- `Output`: The output of a *Candidate Analyzer Module*, are keywords marked as possible candidates for future concepts in the semantic representation, in the form of a ranked list.

Concept Candidate Module The *Concept Candidate Module* is responsible for handling possible candidates for concepts that might be introduced into the semantic representation utilized by the framework. Its main role is to manage possible candidates, and possibly introduce them as an element of the semantic representation, which is commonly known as incremental ontology buildup.

- `Input`: The *Concept Candidate Module* receives concepts in the form of strings (RDF or OWL), with additional context information like ranking as well as usage statistics in a perviously agreed upon XML format.
- `Execution Model`: *Concept Candidate Module* run as a triggered execution initiated by the end of a run from the Concept Candidate Analyzer or an user triggered action, like the submission of new concepts.
- `Output`: The *Concept Candidate Module* stores concept candidates that might be introduced into the semantic representation of the framework into the Concept Candidate indices via the Search Engine Component.

Ontology Editor Module The *Ontology Editor Module* (OEM) is concerned with managing the ontologies in their standard format forms of OWL/RDF representations. It's main purpose is to add, update or delete concepts. A prominent example of a OEM is Protege[3].

- `Input`: The *Ontology Editor* receives concepts and concept operations in a previously agreed upon XML format.
- `Execution Model`: *Ontology Editor Modules* run as a triggered execution.
- `Output`: Added, updated or deleted concepts and concept indices via the Search Engine Component

[3] `http://protege.stanford.edu/`

Operation name	**Description and comments**
Module Lifecycle Management	
Register Module	This operation is used to register an importer module with the component for later use. By registering, a module signals availability for service.
Deregister Module	This operation allows a module to disconnect from the component, and gracefully stop its operations.
Ranking Management	
Submit Ranking	This operation is used to submit a ranking to the User Participation Component
Retrieve Ranking	This operation is used to retrieve ranking from the User Participation Component
Concept Management	
Submit Concept	This operation is used to submit a concept to the User Participation Component
Retrieve Concept	This operation is used to retrieve concepts from the User Participation Component
Submit Concept Association	This operation is used to associate a concept with one or many digital objects
Retrieve Concept Association	This operation is used to retrieve concept associations

Figure 5.37: User Participation Component Interface overview

Interface Description

Component API This interface represents the primary API of the UPC Detailed descriptions of the methods can be found in Figures 5.37 and 5.38.

Parameter name	Type	Direction	Description and comments
Authorization Key	`String`	IN	The module's authorization key
Module Type	`<String, String>`	IN	A key value pair consisting of the type of module being registered, allowed values are `<concept,candidate>` for a Concept Candidate Module, `<concept,editor>` for a Ontology Editor Module, `<concept,analyzer>` for a Candidate Analyzer Module
Module ID	`Integer`	OUT	The handle to the module if it was registered successfully, `null` otherwise

(a) Operation `Register Module`

Parameter name	Type	Direction	Description and comments
Authorization Key	`String`	IN	The module's authorization key
Module ID	`Integer`	IN	The handle to the module as obtained by a call to `registerModule`
Status	`Boolean`	OUT	`true` if module was deregistered successfully, `false` otherwise

(b) Operation `Deregister Module`

Parameter name	Type	Direction	Description and comments
Rank	`Integer`	IN	The rank of a digital concept association.
Type	`String`	IN	The type of the ranking, concept, digital object, social.
Reference GUID	`String`	IN	The id of the ranked entity, being concept, digital object or social entity.
Ranking GUID	`String`	OUT	The id of the ranking.

(c) Operation `Submit Ranking`

Figure 5.38: User Participation Component Interface details

Parameter name	Type	Direction	Description and comments
Ranking GUID	`String`	IN	The id of the ranking.
Ranking	`String<XML>`	OUT	The ranking in an XML-serialized form.

(d) Operation `Retrieve Ranking`

Parameter name	Type	Direction	Description and comments
Concept	`OWL, String`	IN	The concept and its relation to existing concepts as OWL or `string`.
Concept GUID	`String`	OUT	The concept id.

(e) Operation `Submit Concept`

Parameter name	Type	Direction	Description and comments
Concept GUID	`String`	IN	The concept id
Concept	`<String, OWL>`	OUT	A key value pair with the concept GUID and its OWL representation

(f) Operation `Retrieve Concept`

Parameter name	Type	Direction	Description and comments
Concept GUID	`String`	IN	The id of the concept
Digital Object GUID	`String`	IN	The id of the digital object
Concept Association GUID	`String`	IN	The id of the concept association

(g) Operation `Submit Concept association`

Parameter name	Type	Direction	Description and comments
Concept Association GUID	`String`	IN	The concept association GUID
Concept Association	`<String, String>`	OUT	A key value pair of the digital objects GUID and the concepts GUID

(h) Operation `Retrieve Concept association`

Figure 5.38: User Participation Component Interface details (contd.)

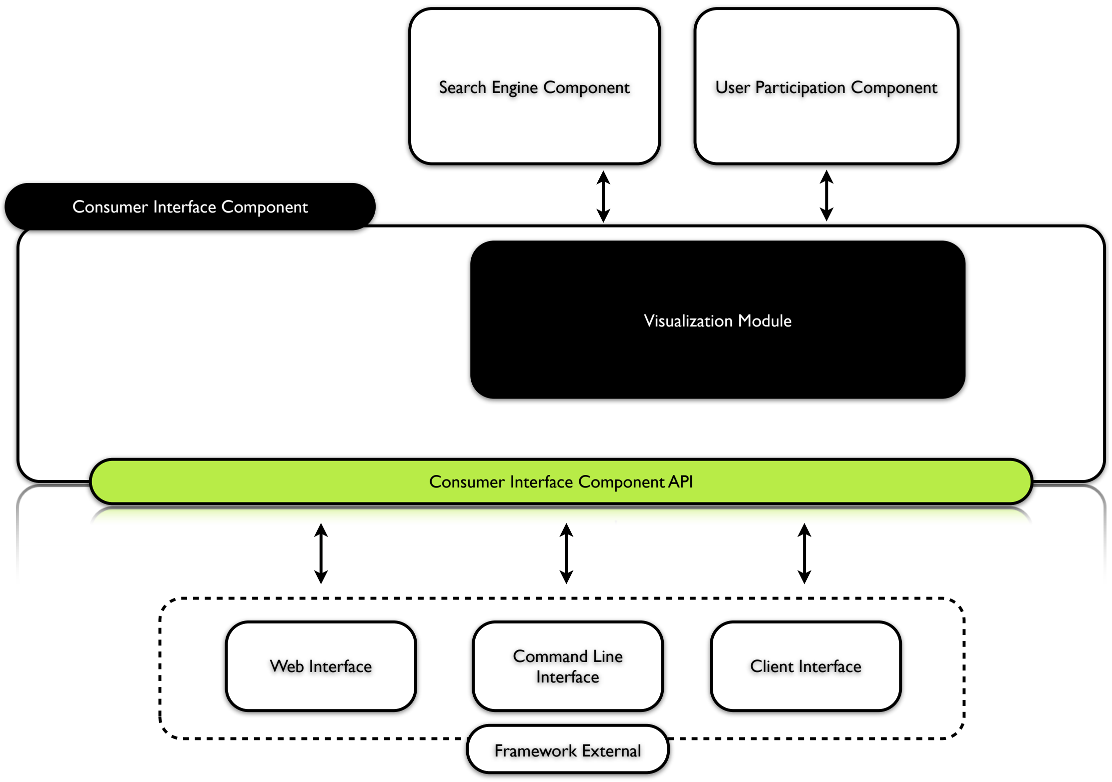


Figure 5.39: Overview of the Consumer Interface Component

5.2.6 *Consumer Interface Component*

The Consumer Interface Component's main role is to act as an abstraction layer for the Framework's functionality. It provides the components' functionality to external participants like Web, Command Line or Client interfaces, via aggregating the APIs of the components. Despite this, it provides the data structures for graphical visualizations through the *Visualization Module*, an overview can be seen in Figure 5.39.

Component Architecture

In the following paragraphs we will describe the principal modules composing the component in greater detail.

Visualization Module The main role of the visualization module is to provide the necessary data structures that can be utilized by visualization algorithms, in order to provide 2D or 3D visualization of concepts and concept associations, as well as search results. Prominent examples are [54], [94], [52],

[63] and [85]. The main structure, these visualizations have in common, are graph based representations of the data. *Nodes* in such a graph are digital objects and/or concepts, *Edges* are relations between digital object, concepts, social entities and digital objects and concepts. Despite this, there are several statistical values that are relevant for the graph, like *Eccentricity*, *Center* and *Transitive Closure*. These values are used to display distances in social or concept networks accurately.

- `Input` The Input of the Visualization Module are Concept, Digital Objects, concept associations and search results from the Search Engine Component and User Participation Component.
- `Execution Model`: The Execution Model is a combined triggered and batched execution. Data aggregation and building the visualization models from the Search Engine Component and User Participation Component can be run as batch jobs, as well as be triggered by an action from a user interface.
- `Output` The Output of a Visualization Model, is a data representation of search results in the graph form introduced previously, concepts and concept to concept, as well as concept to digital object relations, in a previously agreed upon XML-format.

Interface Description

Component API This interface represents the primary API of the Consumer Interface Component UIC Detailed descriptions of the methods can be found in Figures 5.40 through 5.41. Beyond these methods it, as described, aggregates the API methods introduced previously.

Operation name	**Description and comments**
Module Lifecycle Management	
Register Module	This operation is used to register an importer module with the component for later use. By registering, a module signals availability for service.
Deregister Module	This operation allows a module to disconnect from the component, and gracefully stop its operations.
Visualization Management	
Submit Graph	This operation is used to submit a graph to the Consumer Interface Component
Retrieve Graph	This operation is used to retrieve a graph from the Consumer Interface Component
Submit Concept Graph	This operation is used to submit a concept graph to the Consumer Interface Component
Retrieve Concept Graph	This operation is used to retrieve a concept graph from the Consumer Interface Component
Submit Concept Association Graph	This operation is used to submit a concept association graph to the Consumer Interface Component
Retrieve Concept Association Graph	This operation is used to retrieve a concept association graph from the Consumer Interface Component

Figure 5.40: Consumer Interface Component Interface overview

Parameter name	Type	Direction	Description and comments
Authorization Key	`String`	IN	The module's authorization key
Module Type	`<String, String>`	IN	A key value pair consisting of the type of module being registered, allowed values are `<concept,candidate>` for a Concept Candidate Module, `<concept,editor>` for a Ontology Editor Module, `<concept,analyzer>` for a Candidate Analyzer Module
Module ID	`Integer`	OUT	The handle to the module if it was registered successfully, `null` otherwise

(a) Operation `Register Module`

Parameter name	Type	Direction	Description and comments
Authorization Key	`String`	IN	The module's authorization key
Module ID	`Integer`	IN	The handle to the module as obtained by a call to `registerModule`
Status	`Boolean`	OUT	`true` if module was deregistered successfully, `false` otherwise

(b) Operation `Deregister Module`

Parameter name	Type	Direction	Description and comments
Graph	`String<XML>, dot`	IN	The graph of concepts, persons and the relation to digital objects.
Graph GUID	`String`	OUT	The id of the graph.

(c) Operation `Submit Graph`

Parameter name	Type	Direction	Description and comments
Graph GUID	`String`	IN	The id of the ranking.
Graph	`String<XML>, dot`	OUT	The graph of concepts, persons and the relation to digital objects.

(d) Operation `Retrieve Graph`

Figure 5.41: Consumer Interface Component Interface details

Parameter name	**Type**	**Direction**	**Description and comments**
Concept Graph	`String<XML>, dot`	IN	The concept and its relation to existing concepts as graph `string`.
Concept Graph GUID	`String`	OUT	The concept graph id.

(e) Operation `Submit Concept Graph`

Parameter name	**Type**	**Direction**	**Description and comments**
Concept Graph GUID	`String`	IN	The concept graph id
Concept Graph	`String<XML>, dot`	OUT	The concept graph in XML or dot format

(f) Operation `Retrieve Concept Graph`

Parameter name	**Type**	**Direction**	**Description and comments**
Concept Association Graph	`String<XML>, dot`	IN	The concept association graph, showing the associations of concepts to digital objects
Concept Association Graph GUID	`String`	OUT	The id of the concept association graph

(g) Operation `Submit Concept association Graph`

Parameter name	**Type**	**Direction**	**Description and comments**
Concept Association Graph GUID	`String`	IN	The concept association graph GUID
Concept Association Graph	`String<XML>, dot`	OUT	A concept association graph.

(h) Operation `Retrieve Concept association Graph`

Figure 5.41: Consumer Interface Component Interface details (contd.)

5.3 Feasibility of Implementation Analysis

In this section, we will provide guidelines for implementing our proposed concept search framework, give recommendations on realizing the different components, and how to handle issues like scalability, security and performance.

Due to the autonomous nature of the components of the proposed framework, a service-based approach is most suitable for implementing the identified parts of the solution. We suggest the creation of the following decoupled independent services:

- *Digital Object Importer Service*: Responsible for providing the system with a canonical representation of every imported digital object.
- *Semantic Extraction Service*: Extracts relevant information from imported digital objects, and fills the system's indices.
- *Search Engine Service*: Performs the actual search operations, as requested by user queries, aggregates and ranks results from the system's indices.
- *User Participation Service*: Allows for the extension of the system's semantic models by means of feedback mechanisms.
- *Consumer Interface Service*: Provides an aggregated abstraction layer for user interfaces to access all parts of the search framework.

In small-scale implementations, the *Digital Object Importer Service* could also be realized as part of the *Semantic Extraction Service*. Furthermore, the *Search Engine*, *User Participation*, and *Consumer Interface* services could be combined into one service, to minimize deployment complexity, at the expense of some flexibility in terms of scalability and distribution.

The services can be implemented using one or more of many available technologies and programming languages. Notable programming languages include Java [10], C♯ [38], and Ruby [73]. Several frameworks for implementing services are available for each of the mentioned languages, as seen below. When using Java, services can be built using the *Spring Framework* [100], either as full-blown web services using *Spring Web Service* [2], or as XML-RPC service [29; 18; 82]. When using C♯, the most prominent framework for building web services is ASP.NET [13; 96; 70]. Finally, when using Ruby, restful services [80] can be implemented using Ruby on Rails [97; 72].

The choice of technology, however, is mainly a matter of preference (or existing infrastructure), as the mentioned technologies are all perfectly capable of providing the tools necessary to implement the proposed system, and the architecture even allows for mixing technologies to be able to use the best tool for each job.

— ❧ —

In the following sections we will outline special considerations and implementation suggestions for each part of the proposed architecture:

5.3.1 Digital Object Importer Service

The *Digital Object Importer Service* is a simple, single-task service, responsible for maintaining object identity. To perform this task, the service must provide *Importer Modules* with a persistent datastore to hold object states, in order to minimize multiple successive imports of the same digital object.

The digital object state store can be realized using a database system, whereby both, RDBMS, as well as document-oriented systems, are suitable.

Importer modules can either be designed to run in the scope of the service, or — for an even more flexible, if not as tightly controllable approach — be realized as service clients, accessing the importer service only via its API, thus posing absolutely no limits on the implementation language and used technologies, except from the ability to provide digital objects via the provided API.

5.3.2 Semantic Extraction Service

The *Semantic Extraction Service* is responsible for extracting the required information from the imported digital objects. This is service much more complex than the importer, it is therefore not feasible to create all functionality from scratch. A viable approach would thus rely on known and proven tools, such as GATE, to perform the extraction tasks. The GATE toolkit is a powerful text engineering framework, and also includes an information extraction component, called ANNIE (A Nearly New Information Extraction System). The ANNIE system provides a multitude of components for information extraction and named entity recognition, such as tokenizers, lemmatizers, gazetteers and sentence splitters. Since the focus of the service at hand is semantic extraction, the

OntoGazetteer is of special interest for us, as it provides means for identifying named entities using gazetteer lists linked to ontological concepts. An issue not tackled by the GATE toolkit is the generation of gazetteer lists for matching ontological concepts. It would be desirable to build a tool capable of automatically generating the necessary lists from the ontology, possibly using services such as WordNet[4] to incorporate synonyms.

There are several approaches to integrate a GATE/ANNIE application in the semantic extraction service. One approach is the in-process execution of the module, restricting the implementation language to that of GATE, namely Java. A simpler approach would be to use a simple configurable wrapper application handling the execution of the GATE application, which is executed by a wrapper module (which in turn can either run in the service process or out-of-process), that is responsible for the conversion of the application output into an RDF format suitable for the service.

After the extraction stage is complete, the found concept annotations need to be stored in an optimized RDF store for efficient subsequent retrieval. Notable examples are Sesame [21], AllegroGraph[5], Jena [74], and Virtuoso[6], each of which are capable of efficiently storing and retrieving RDF data.

The *Semantic Extraction Service* is also responsible for extracting keywords from the imported digital objects. For this problem, it is also recommended to use well known and proven tools such as *Lucene*[7], *Lemur*[8] or *Apache Solr*[9], as they perform their task in an optimized manner, and have been in use for a long time (Lucene: 10 years, Lemur: 6 years).

For larger deployments it is important that the used tools implement a scaling strategy to cope with rising loads. Creating distributed indices using Lucene is supported by a number of tools, such as *Katta*[10], *Apache Solr*, and *Distributed Lucene* [22]. These tools allow for replication and sharding of indices to balance load and increase performance. *Katta* and *Distributed Lucene* use *Apache Hadoop*[11] for the distributed processing of data, an implementation of the Map/Reduce approach.

[4] http://wordnet.princeton.edu/
[5] http://agraph.franz.com/
[6] http://virtuoso.openlinksw.com/
[7] http://lucene.apache.org/
[8] http://lemurproject.org/lemur.php
[9] http://lucene.apache.org/solr/
[10] http://katta.sourceforge.net/
[11] http://hadoop.apache.org

5.3.3 Search Engine Service

The search engine service is responsible for maintaining an up-to-date collection of the available indices, in order to be able to distribute incoming queries to achieve maximum performance.

As discussed in the section above, there are several approaches to distribute indices. The Semantic Extraction Service and the Search Engine Service both access the indices, hence they have to support the same set of technologies. This dependency, however, can mostly be encapsulated in an *Index Manager Module*, which in turn can be shared by both services, in order to reduce maintenance overhead and code duplication.

Since the service will eventually send queries to multiple distinct indices, it is responsible for combining the returned result sets, and ranking them according to query priorities. Finally, the service shall return the aggregated result set to the caller that provided the query.

To further support queries based on the combined semantic representation (5.1) the Search Engine Service needs to be able to support keyword and conceptual queries. In order to do so it needs to analyze whether a submitted query is keyword or concept, and forward it to the according modules. For Keyword based search the most viable tools are Lucene and Solr, which is a search server on top of Lucene, that exposes its functionality via a Service interface. and therefore fulfills the prior outlined requirements for the Search Engine Service. For the concept search, which covers the semantic and social parts of the combined representation, the suggested approach is to utilize Jena or Sesame, both providing service based interaction, as well as rich query support via SPARQL, including the necessary support for distributed indices.

5.3.4 User Participation Service

The User Participation Service handles all forms of ranking and feedback, as well as extensions of the concepts, either by allowing editing of the utilized ontologies, or maintaining concept candidates, that will be later introduced as part of the ontology, or as a new ontology. As we outlined previously, user participation plays a vital role, but is also a time consuming task and therefore often neglected by the users. To overcome this limitation, the game based approaches by [98], as well as the use of Captchas in the Authentication phase [6] provide feasible approaches to solve this problem. For the process of augmenting concepts, the most feasible approach is Tagging, which has been shown to provide a sufficiently accurate means for incremental ontology buildup [53].

5.3.5 Consumer Interface Service

The main role of the Consumer Interface Service is to provide the functionality of the Framework to external User Interface implementations by aggregating the previously described APIs. Additionally to that, it can supply libraries to speed up client development. Popular examples would be JavaScript for web clients, as well as libraries for Java, C♯ and Ruby. Despite these basic features, the *Visualization Module* can provide the necessary graph structures for 2D visualizations to simplify the process of displaying concepts, concept associations as well as social associations. The most reasonable approach, despite providing basic graph structures via JSON, is to utilize the popular DOT format by GraphViz[12].

Additionally, the Consumer Interface Service is responsible for transparently accessing multiple distributed instances of underlying services, as well as to allow for multiple instances of this service, if it becomes a performance bottleneck.

5.3.6 Nonfunctional considerations

The fact that the proposed architecture is purely component oriented enables the easy introduction of aspects such as *Authentication* and *Authorization*.

The flexible nature of the architecture furthermore allows for realizing highly performant and scalable systems, by parallelizing critical parts of the process. The extraction processes can be easily executed in parallel, given that the document corpus to be imported can be split according to available processing resources. The used indices can be distributed by means of replication or sharding, providing advantages in subsequent query operations, as the search process can be realized using a Map/Reduce approach such as *Apache Hadoop*, to optimize performance.

[12] http://www.graphviz.org/

5.4 Evaluation of the Semantic Search Framework

In this section we evaluate the previously outlined framework according to the introduced requirements shown in Section 4.3 on a conceptual level.

5.4.1 *Functional Evaluation*

In terms of Functional Requirements the evaluation concentrates on general aspects, since specific fulfillment of the evaluation condition is highly dependent upon the implementation. The proposed **Digital Object Importer Component** with its exchangeable and extensible *Digital Object Importer Module* enables the support for multiple *Digital Object Types and Formats* despite this it delegates concerns of *Digital Object Size*, *Number of Digital Objects* as well as *Digital Object update cycle* to the relevant systems. The **Semantic Extraction Component** provides to efficiently integrate different Extraction Mechanisms, clearly specifying the use of *Standard formats* like RDF and OWL in its interface description. Further *User Participation* and *Incremental Ontology Buildup* means are provided by the **User Participation Component** which also covers the concerns in terms of *Semantic Annotation Mechanisms*. Despite the capabilities of the **Digital Object Importer Component** to incorporate a variety of Stores and Sources for the Framework all Components provide a clear and simple API that allows for easy *Incorporation of existing systems*. The *Index Manager Module* in the **Search Engine Component** enables the *Integration of existing indices* and therefore also Path-Lookup and Navigational Indices in terms of *Index Model support*. The modular setup of the **Search Engine Component** further enables different *Search Capabilities* through integration of different *Search Modules* and according *Result Aggregation*. This also is the fundamental for *Combined Search*. The **Consumer Interface Component** concludes the Framework by aggregating all functionality and additionally provides *Visualization* capabilities through its *Visualization Module*.

5.4.2 *Nonfunctional, Software Engineering and Deployment Evaluation*

The highly modular design of our framework allows, as explained previously (see 5.2), the integration of different implementations for the components and modules. Furthermore, it allows for multiple distribution scenarios (see Section 5.2.1), making it a perfect fit for a plethora of deployment scenarios. Therefore, it supports concrete implementations to achieve the *Scalability* by selecting the right deployment scenario. The fine granularity of the framework allows to use the tools that are suited best for the specific step of the search process and, therefore, it supports concrete implementations to achieve the *Performance* and *Accuracy* by selecting the best possible tools. The service based nature of the framework as well as the detailed API ensures *Reusability*, *Extensibility* and *Interoperability*.

6 *Prototype and Evaluation*

6.1 Overview

In order to demonstrate the proposed concept search framework, we need to conduct a concept demonstration. In this thesis, the concept demonstration includes:

- a small-scale of the design of the proposed concept search
- functional and nonfunctional analysis of the prototype.

We have decided to use the Space domain as representative specialized domain to test our concept demonstration.

6.2 Prototype of The Proposed Concept Search Framework

6.2.1 *General Architecture*

The developed prototype takes the recommendations from the previous sections into account. Our prototype has included the following components:

- Digital Object Importer and Semantic Extraction Component
- Search Engine Service
- Consumer Interface Service

Our service access method of choice is REST, as it provides for low maintenance overhead, and is supported by all relevant technologies. An overview of the implemented components and their interactions according to our proposed conceptual architecture can be seen in Figure 6.1.

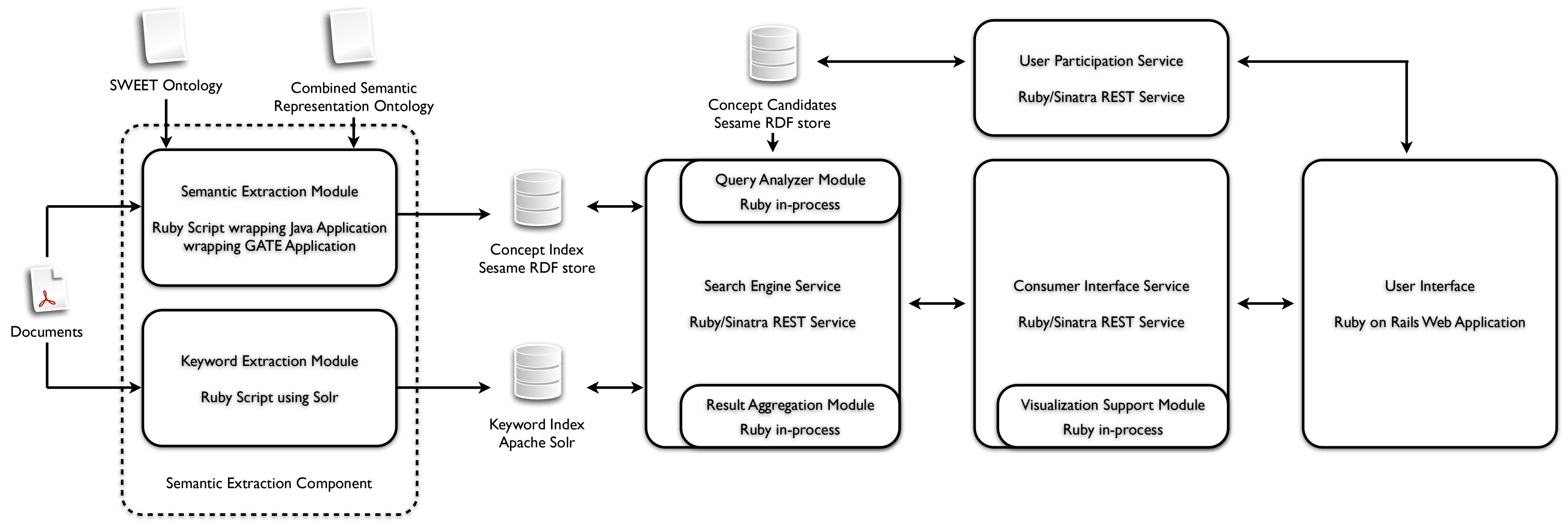

SWEET Ontology
Combined Semantic Representation Ontology
Documents
Semantic Extraction Module
Ruby Script wrapping Java Application wrapping GATE Application
Keyword Extraction Module
Ruby Script using Solr
Semantic Extraction Component
Concept Index
Sesame RDF store
Keyword Index
Apache Solr
Concept Candidates
Sesame RDF store
Query Analyzer Module
Ruby in-process
Search Engine Service
Ruby/Sinatra REST Service
Result Aggregation Module
Ruby in-process
User Participation Service
Ruby/Sinatra REST Service
Consumer Interface Service
Ruby/Sinatra REST Service
Visualization Support Module
Ruby in-process
User Interface
Ruby on Rails Web Application

Figure 6.1: Concept Demonstration Architecture Overview

6.2.2 *Linked Data Model*

The combined semantic representation, proposed in Section 5.1 has been fully implemented in an ontology (see Listing B.1), enabling the desired linking of digital object, human, semantic (domain specific ontologies), and social concepts. Also, the SWEET ontology has been integrated into the data model to represent domain-specific concepts of our test domain. For the purposes of this prototype, we are integrating only subclasses from the `PhysicalPhenomena` and `PlanetaryRealm` classes. The data extracted from digital objects is represented as RDF triples, and persisted in a Sesame datastore.

6.2.3 *Semantic Extraction*

In our prototype, the semantic extraction component is an Executable, following the Batch Execution model for the sake of simplicity, and includes two modules: *Semantic Extraction Module*, and *Keyword Extraction Module*.

The *Semantic Extraction Module* acts as a wrapper around a GATE application, processing documents in batches. The instantiated GATE app uses the ANNIE information extraction system to perform named entity recognition using an *OntoGazetteer* module. To obtain a gazetteer list for the named entity recognition to be performed, we implemented an automatic gazetteer list generator, transforming class names from the ontology into free text representations. Within the *Semantic Extraction Module*, the batch-processing wrapper module is a Java application, accepting a saved GATE application and a set of documents as parameters. An exemplary invocation is shown in Listing 6.1.

```
$ java BatchProcessApp -g gateApp.gapp document1.pdf document2.doc
```

Listing 6.1: Exemplary Semantic Extraction Module invocation

The application loads the provided GATE application and processes each of the passed documents, as seen in Listing 6.2.

```
Corpus corpus = Factory.newCorpus("BatchProcessApp␣Corpus");
application.setCorpus(corpus);

for(int i = firstFile; i < args.length; i++) {
  File idFile = new File(args[i]);
  File docFile = new File(workingDir + "/" +
                  new BufferedReader(new FileReader(idFile)).↩
                      readLine());
  Document doc = null;
```

```
    try {
      doc = Factory.newDocument(docFile.toURI().toURL(), encoding);
    } catch(Exception e) {
      continue;
    }
    corpus.add(doc);
    application.execute();
  }
```

Listing 6.2: Simplified internal process of the Semantic Extraction Module

The GATE application uses the automatic gazetteer list generator to provide a set of gazetteer lists for named entity recognition. The annotations generated by the application are then stored in the attached *Sesame* RDF store, as shown in Listing 6.3.

```
    for(Annotation a:doc.getAnnotations("onto")) {
      URI concept = f.createURI(a.getFeatures().get("ontology")+"#"+a↩
          .getFeatures().get("class"));
      URI document = f.createURI(docFile.getCanonicalFile().toURI().↩
          toURL().toString());
      URI topic = f.createURI("http://xmlns.com/foaf/0.1/primaryTopic↩
          ");

      try {
        RepositoryConnection con = myRepository.getConnection();

        try {
          con.add(document, topic, concept);
        } finally {
          con.close();
        }
      } catch (OpenRDFException e) {
      }
    }
```

Listing 6.3: Simplified internal process of the Semantic Extraction Module

The used GATE application is shown in its entirety in Listing B.4. The complete code of the wrapper module storing the extracted annotations, can be found in Listing B.3, and the automatic gazetteer list generator can be found in Listing B.2.

The *Keyword Extraction Module* acts as a wrapper around *Apache Solr*, passing imported documents to the Solr's keyword extractor for further processing.

6.2.4 Social Semantic Search

The *Social Semantic Search Service* is implemented using Ruby and the Sinatra[1] framework. It provides a REST service API. The service analyzes incoming queries and performs the search operations on the attached indices. Keyword queries are sent to the Lucene index, semantic and social queries are executed using SPARQL against the Sesame RDF store. The principal interface of the *Search Module* is shown in Listing 6.4.

```
get '/documents/search/:query'
get %r{^/tags(?:\.([\w]+))?} |format|
```

Listing 6.4: Principal methods of *Search Module* API

6.2.5 Result aggregation and presentation

The results returned from the Lucene keyword index and the Sesame RDF store are aggregated by only using documents contained in all result sets, ranked according to the quality value from the keyword search engine. This approach is only possible if the submitted query contains a semantic, as well as a keyword query.

If the submitted query is purely keyword-based, the search service only needs to consult the keyword index, and no result aggregation is performed. Likewise, if the submitted query is purely semantic, the search service only needs to consult the RDF store, and no result aggregation is performed.

6.2.6 User Participation/Incremental Ontology Buildup

The incremental ontology buildup component is realized as a Ruby/Sinatra-based RESTful service. It stores concept candidate annotations derived from user tags in the concept candidate Sesame RDF store.

6.3 Experiments

In this section we describe the experiments that were performed with the Prototype, as well as the environment and the used test data.

[1]http://www.sinatrarb.com

6.3.1 *Document Corpora*

The documents used for the experiments were extracted from the ESA general studies programme[2]. We used only Executive Summaries.

This source provided 263 documents as pdf, doc and excel documents that were used for the initial testing of the framework. Several documents had to be neglected as their information was not extractable due to Digital Right restrictions. In order to simulate payload and test the specified performance criteria established in Chapter 4, it was necessary to extend the amount of documents. We did so by introducing a so called payload simulator (see B.6), that was introduced in the document importer process and generated two Document corpora based upon the 263 documents extracted from the site specified above.

Corpus A containing 1000 duplicated documents from the set.

Corpus B containing 25000 duplicated documents from the set.

The corpora included documents in all specified formats (pdf, doc, excel), images and videos were not available and are skipped in our experiments since the extraction process only depends on the specified metadata and is therefore similar to documents. Certain documents had to be neglected due to the fact that their information was marked as non extractable.

6.3.2 *Functional Experiments*

In this section we demonstrate the functional capabilities of our prototype, the focus lies on demonstrating the capabilities in respect of semantic extraction, semantic queries as well as the search and result presentation capabilities in general.

Semantic Extraction

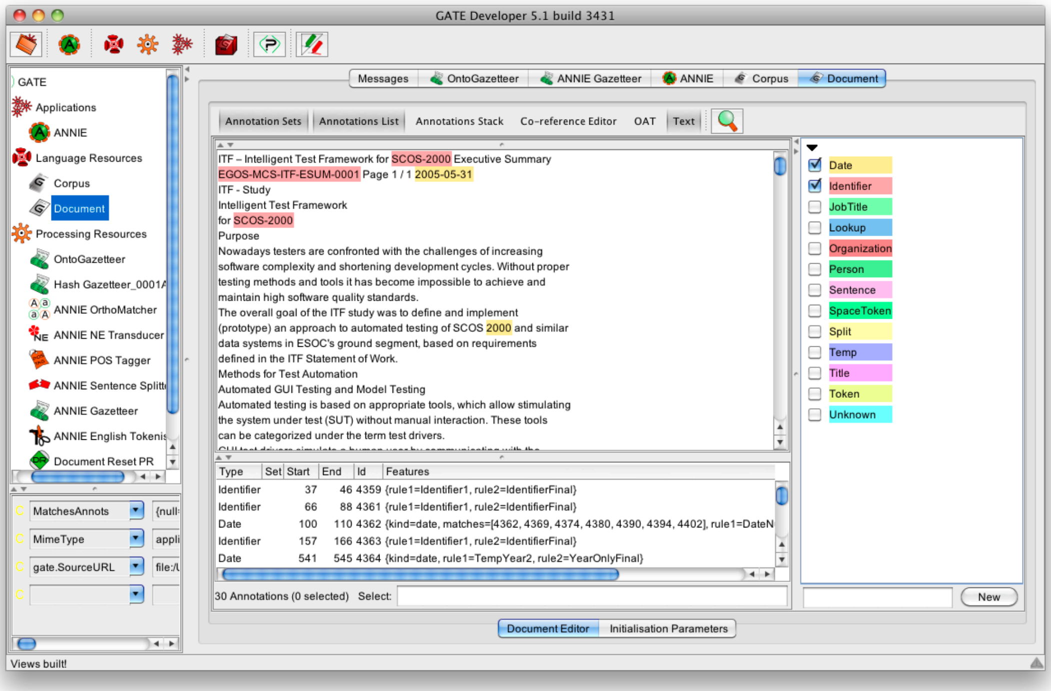
We utilize GATE in our prototype to perform the semantic extraction from the previously defined Document corpus. GATE provides means for automatic as well manual annotation.

For automatic semantic annotation we utilize the automatically generated lists described in Section 6.2.3. This greatly reduces the time spent usually

[2]http://www.esa.int/SPECIALS/GSP/

to generate this files and therefore significantly improves the value of GATE as a semantic extraction facility. Furthermore, the distribution of components provides massive performance enhancements in terms of extraction shown in Section 6.3.3.

Despite this GATE gives the opportunity to manually annotate via the GATE GUI shown in Figure 6.2



ITF – Intelligent Test Framework for SCOS-2000 Executive Summary
EGOS-MCS-ITF-ESUM-0001 Page 1 / 1 2005-05-31
ITF - Study
Intelligent Test Framework
for SCOS-2000
Purpose
Nowadays testers are confronted with the challenges of increasing
software complexity and shortening development cycles. Without proper
testing methods and tools it has become impossible to achieve and
maintain high software quality standards.
The overall goal of the ITF study was to define and implement
(prototype) an approach to automated testing of SCOS 2000 and similar
data systems in ESOC's ground segment, based on requirements
defined in the ITF Statement of Work.
Methods for Test Automation
Automated GUI Testing and Model Testing
Automated testing is based on appropriate tools, which allow stimulating
the system under test (SUT) without manual interaction. These tools
can be categorized under the term test drivers.

Type	Set	Start	End	Id	Features
Identifier		37	46	4359	{rule1=Identifier1, rule2=IdentifierFinal}
Identifier		66	88	4361	{rule1=Identifier1, rule2=IdentifierFinal}
Date		100	110	4362	{kind=date, matches=[4362, 4369, 4374, 4380, 4390, 4394, 4402], rule1=DateN
Identifier		157	166	4363	{rule1=Identifier1, rule2=IdentifierFinal}
Date		541	545	4364	{kind=date, rule1=TempYear2, rule2=YearOnlyFinal}

30 Annotations (0 selected) Select:

Figure 6.2: GATE annotation view

This ensures great flexibility for the semantic annotation process. If GATE isn't suitable for the specific task, or a newer version should be integrated, this is also easily manageable through our modular architecture.

Semantic Queries

Our framework allows the utilization of powerful semantic queries beyond simple keyword search to retrieve concepts, inferred concepts as well as related documents. This enables more expressible search, for example when searching for a concept you also retrieve the documents associated with inferred concepts, therefore providing more accurate results. Sample Queries for retrieving the domain specific ontology hierarchy as well as retrieving inferred concepts can be seen in Listings 6.5 and 6.6 respectively.

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX phys:<http://sweet.jpl.nasa.gov/2.0/phys.owl#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
PREFIX astroPlanet:<http://sweet.jpl.nasa.gov/2.0/astroPlanet.owl↩
    #>
PREFIX sesame:<http://www.openrdf.org/schema/sesame#>

SELECT DISTINCT ?class ?directsuperclass WHERE {
  ?class sesame:directSubClassOf ?directsuperclass .
  { ?directsuperclass rdfs:subClassOf astroPlanet:PlanetaryRealm ↩
      } UNION
  { ?directsuperclass rdfs:subClassOf phys:PhysicalPhenomena }
}
```

Listing 6.5: Exemplary SPARQL Query to retrieve all concepts and their superclasses

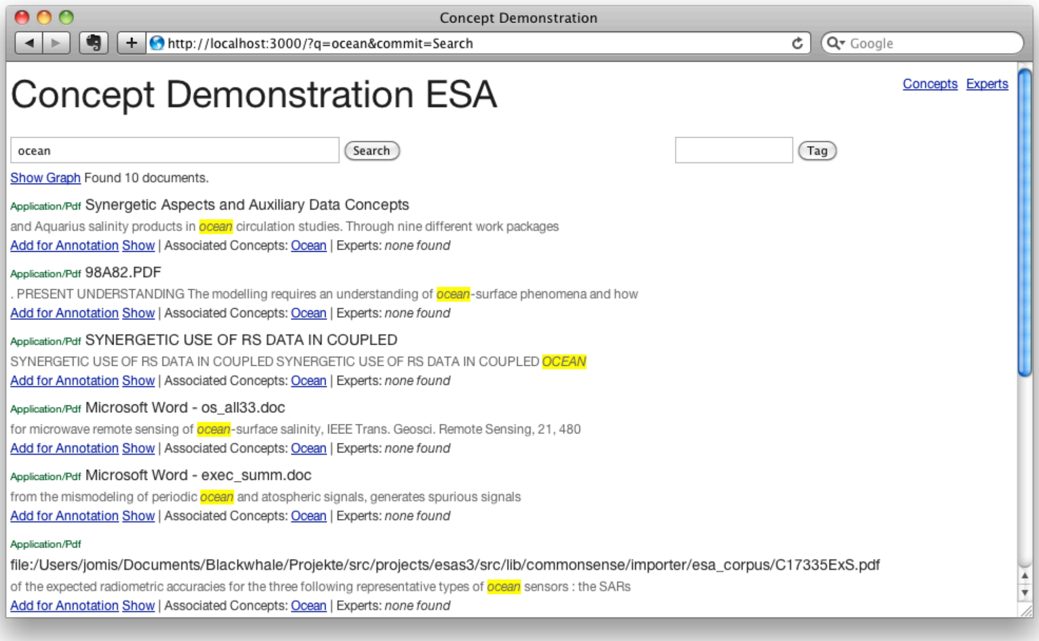
```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX phys:<http://sweet.jpl.nasa.gov/2.0/phys.owl#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
PREFIX astroPlanet:<http://sweet.jpl.nasa.gov/2.0/astroPlanet.owl↩
    #>
PREFIX sesame:<http://www.openrdf.org/schema/sesame#>

SELECT DISTINCT ?class WHERE { ?class rdfs:subClassOf
phys:PhysicalPhenomena . ?class rdf:type owl:Class }
```

Listing 6.6: Exemplary SPARQL Query to retrieve all subclasses for a specific concept

Search Interface

Beyond the previously outlined usages, the main benefit lies in the sophisticated results the framework can provide. Figure 6.3 shows the results of a combined search for the concept "Ocean". The combined data model, as well as the expressive query support enables a far more detailed result presentation, showing not only the documents where the keyword was found, but also documents that were annotated with the concept "Ocean". Beyond this, the result also shows associated experts for the document and the concept, enabling expert based faceted browsing. Results can be simply annotated via the Tag

Input seen on the right, which shows an intuitive and easy implementation of the User Participation concept.



Concept Demonstration ESA

ocean

Show Graph Found 10 documents.

Application/Pdf Synergetic Aspects and Auxiliary Data Concepts
and Aquarius salinity products in ocean circulation studies. Through nine different work packages
Add for Annotation Show | Associated Concepts: Ocean | Experts: *none found*

Application/Pdf 98A82.PDF
. PRESENT UNDERSTANDING The modelling requires an understanding of ocean-surface phenomena and how
Add for Annotation Show | Associated Concepts: Ocean | Experts: *none found*

Application/Pdf SYNERGETIC USE OF RS DATA IN COUPLED
SYNERGETIC USE OF RS DATA IN COUPLED SYNERGETIC USE OF RS DATA IN COUPLED OCEAN
Add for Annotation Show | Associated Concepts: Ocean | Experts: *none found*

Application/Pdf Microsoft Word - os_all33.doc
for microwave remote sensing of ocean-surface salinity, IEEE Trans. Geosci. Remote Sensing, 21, 480
Add for Annotation Show | Associated Concepts: Ocean | Experts: *none found*

Application/Pdf Microsoft Word - exec_summ.doc
from the mismodeling of periodic ocean and atospheric signals, generates spurious signals
Add for Annotation Show | Associated Concepts: Ocean | Experts: *none found*

Application/Pdf
file:/Users/jomis/Documents/Blackwhale/Projekte/src/projects/esas3/src/lib/commonsense/importer/esa_corpus/C17335ExS.pdf
of the expected radiometric accuracies for the three following representative types of ocean sensors : the SARs
Add for Annotation Show | Associated Concepts: Ocean | Experts: *none found*

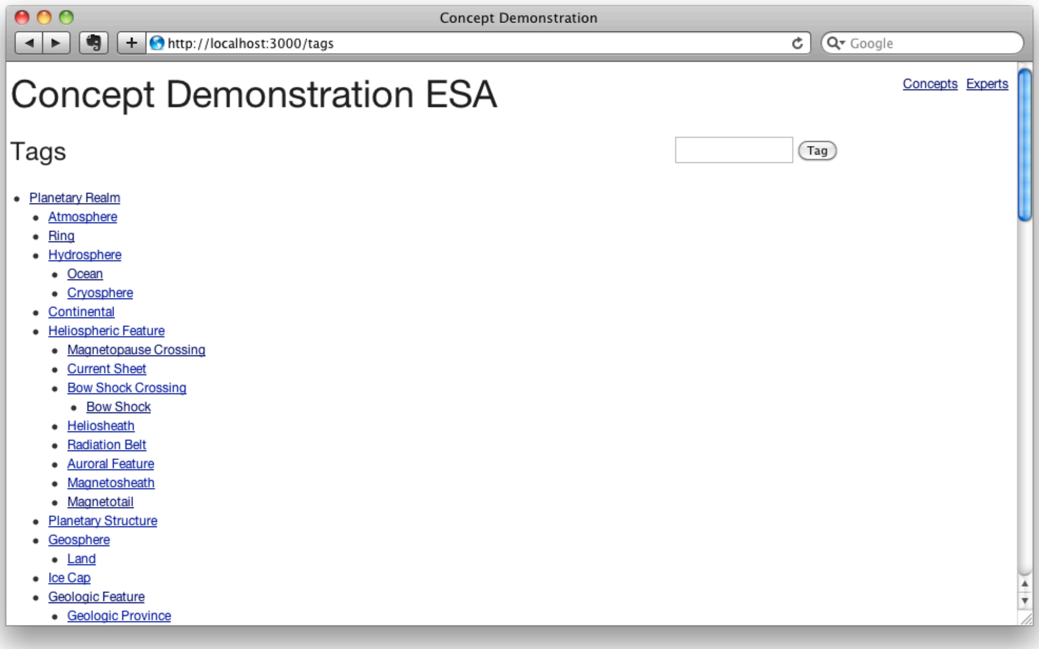
Figure 6.3: Results for a combined Search for the keyword and concept "Ocean"

In addition to keyword based concept search, the Concept Demonstration also provides easy faceted browsing, seen in Figure 6.4. The screenshot shows a hierarchical listing of extracted concepts, clicking the link leads the user to the associated documents, and is also capable of displaying documents associated with inferred concepts; this is done via the Semantic Query capability outlined above.

Despite the faceted browsing capability, it is also easy to provide additional visualizations for concepts and related documents via the Visualization Component, shown in Figure 6.5. The screenshot shows a directed graph rendered using Graph.js[3] of the concepts that allows for easy browsing of concepts and associated documents via a common visual paradigm. Figure 6.6 shows a 3-dimensional fully interactive representation.

All the outlined capabilities are easily exchangeable through our modular architecture and therefore enable superior extendability over existing approaches.

[3]http://ajaxian.com/archives/new-javascriptcanvas-graph-library



Concept Demonstration ESA

Tags

- Planetary Realm
  - Atmosphere
  - Ring
  - Hydrosphere
    - Ocean
    - Cryosphere
  - Continental
  - Heliospheric Feature
    - Magnetopause Crossing
    - Current Sheet
    - Bow Shock Crossing
      - Bow Shock
    - Heliosheath
    - Radiation Belt
    - Auroral Feature
    - Magnetosheath
    - Magnetotail
  - Planetary Structure
  - Geosphere
    - Land
  - Ice Cap
  - Geologic Feature
    - Geologic Province

Figure 6.4: Faceted Browsing domain hierarchy overview

6.3.3 *Non-functional Experiments*

In this section we test the performance of our Concept Demonstration in terms of extraction and retrieval/search speeds. The criteria of precision and recall were neglected due to the fact that it depends on the quality of the annotations provided for GATE, which have not been the main focus of this thesis.

6.3.4 *Environment*

The experiments were conducted both on a single machine as well as on a XGrid Cluster.

The single machine utilized in the experiments was a MacBook Pro 2007 model with the following specifications:

Processor: 2.4 GHz Intel Core 2 Duo

Cores 2 (2 Processes in parallel)

RAM: 4GB 667 MHz DDR2 SDRAM

The cluster used in the experiments was an Apple XGrid Cluster with 4 cores and the following specifications.

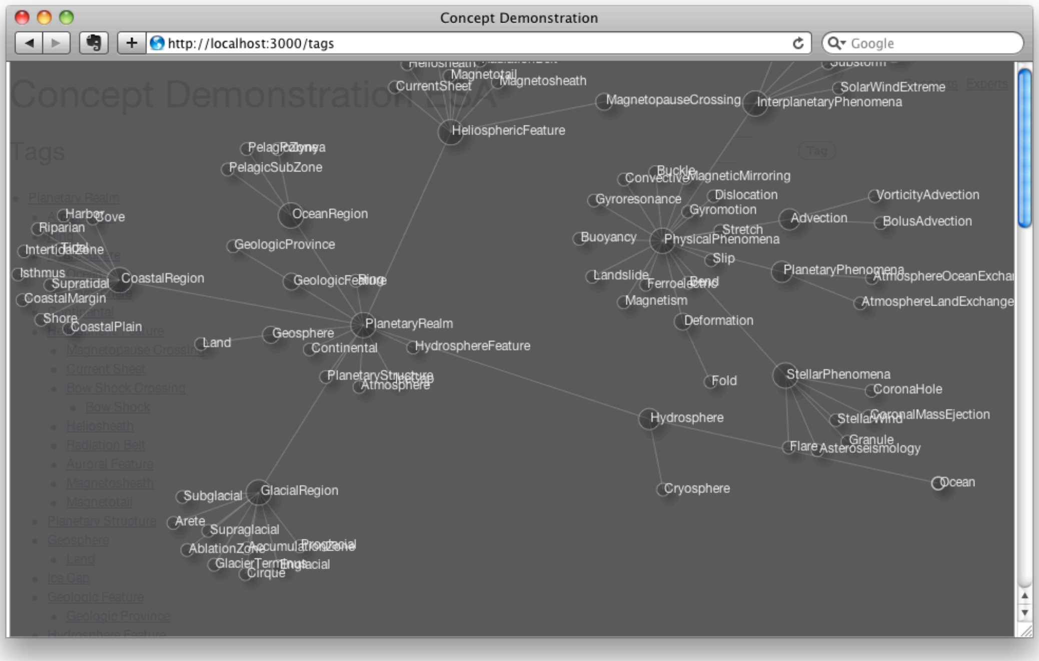


Figure 6.5: Browsable graph representation of the domain specific ontology

Processor Machine 1: 2.4 GHz Intel Core 2 Duo

Processor Machine 2: 2.66 GHz Intel Core 2 Duo

Cores 4 (4 Processes in parallel)

RAM Machine 1: 4GB 667 MHz DDR2 SDRAM

RAM Machine 2: 4GB 1067 MHz DDR3 SDRAM

Nodes were connected via an 100Mbps Ethernet connection. The system setup for extraction is shown in Figure 6.7 and Figure 6.8 shows the setup for the search experiments.

6.3.5 *Extraction*

For the semantic extraction, we simulated with four different chunk sizes processing 50, 100, 250 and 500 documents in a single batch, each batch being executed sequentially. Those chunk sizes were run on the single machine, and on the cluster in parallel both with 1000 and 25000 documents. Each test was conducted three times and the weighted average was recorded. Figure 6.9

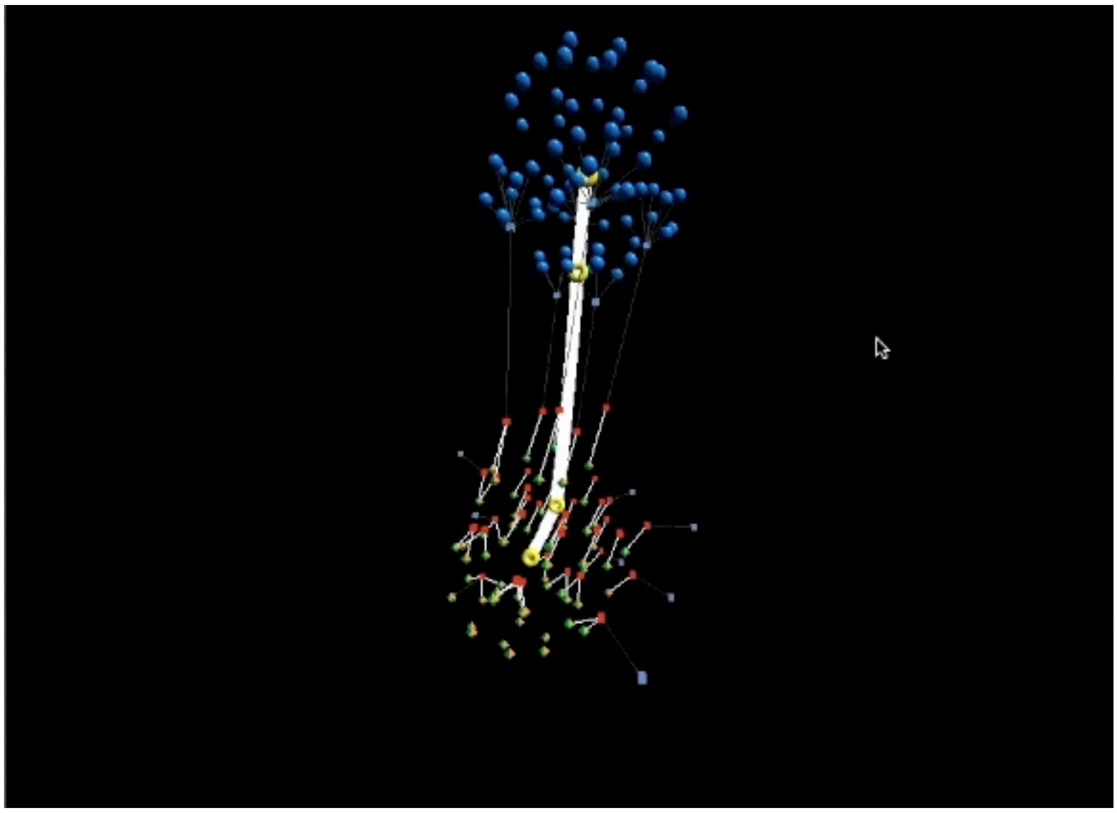


Figure 6.6: Browsable 3 dimensional representation of the domain specific information

and Figure 6.10 show the graph for the extraction times in seconds with 1000 documents on a single machine, as well as on the cluster. Bigger chunk sizes tend to deliver better results, due to the fact that the GATE process overhead is minimized and reaches a saturation level, depending on the available Memory. This stays true for distributed execution, where significant performance boosts could be achieved.

To simulate a more significant payload we also ran the experiments with 25000 documents, getting the same result trend shown previously, the results can be seen in Figure 6.11 and Figure 6.12. The experiments clearly show the massive performance gains that can be achieved by utilizing the distribution of components our architecture allows. Since the extraction and indexing times are growing linearly, it further shows a very good performance trend for handling massive document corpora.

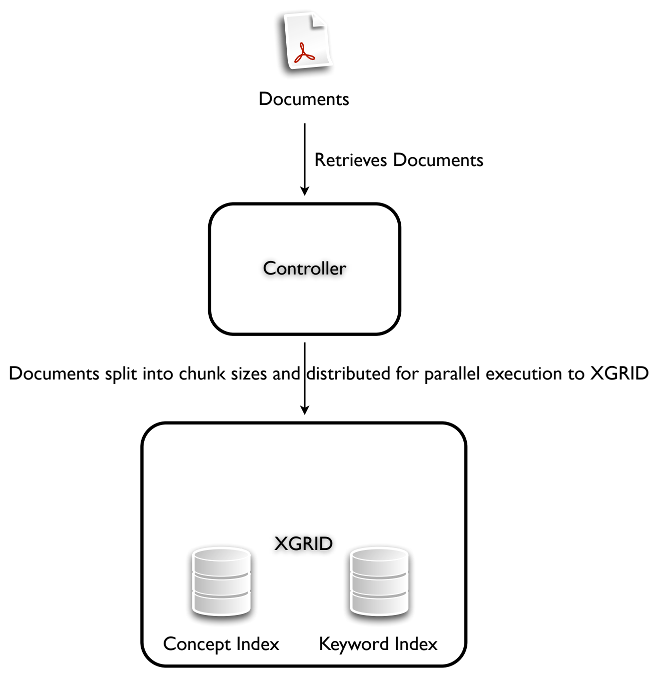

Documents
Retrieves Documents
Controller
Documents split into chunk sizes and distributed for parallel execution to XGRID
XGRID
Concept Index
Keyword Index

Figure 6.7: Setup of the test system for extraction

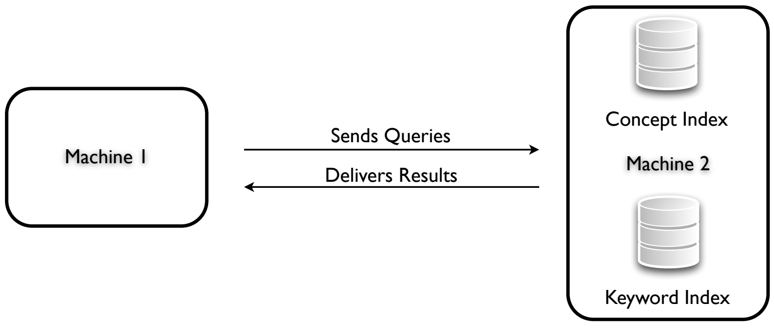


Figure 6.8: Setup of the test system for search)

6.3.6 *Search and Retrieval*

To test the retrieval/search speed of the Concept Demonstration, we utilized the two nodes of the previously described XGrid Cluster. *Node One* executing the query, *Node Two* performing the search and sending the results back to *Node One*, in order to respect the network delays as well. For the retrieval tests, several different keyword and concept queries were performed, each of them 3 times and again the weighted average was recorded. Each query was measured with different concurrency levels simulating different load patterns.

Figure 6.13 shows the results for the Keyword Query. Since plain keyword queries were not the main focus, we used an off the shelf Solr implementa-

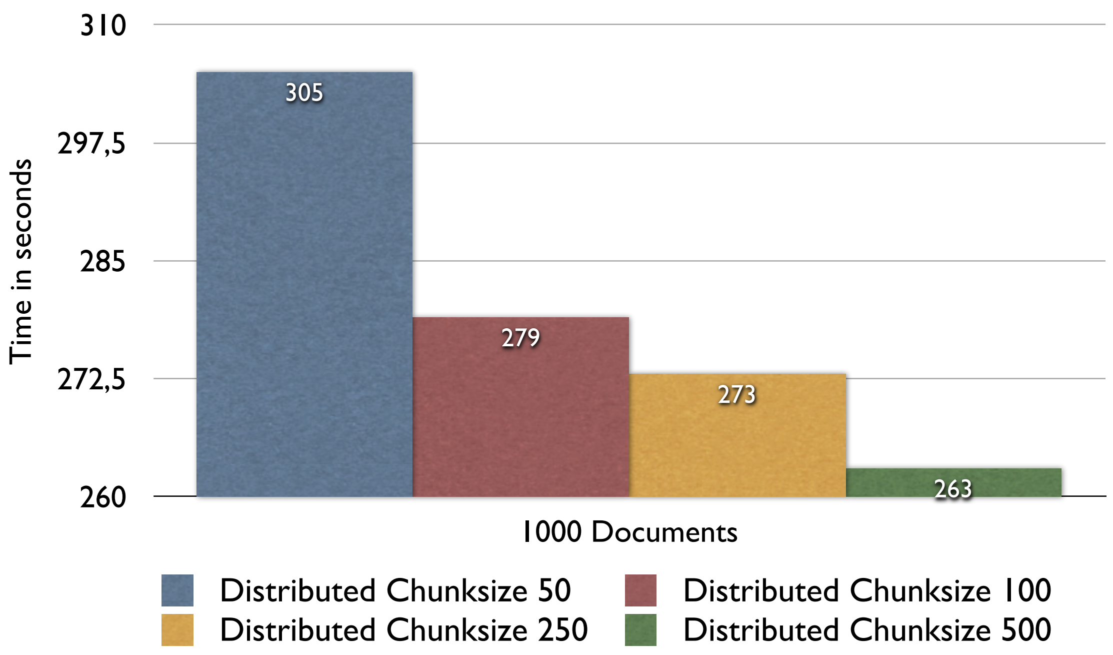


Figure 6.9: Keyword and Semantic Extraction Time in seconds for 1000 documents

tion, with no performance optimizations, hence the rather slow response time results for plain keyword search. However, this poses no problem since our framework design allows for easy replacement of the specific implementation, and therefore, for example, a Google Search Appliance integration would provide way better results in terms of response time.

Figure 6.14 shows the results for the Concept Query.

6.4 Comparison with Existing Systems

The digital library systems outlined in Section 3.9 have not been explicitly designed or tested for specialized domains. From the design's point of view, our proposed concept search framework has several features that other systems do not cover. Furthermore, our framework is specially designed to be able to adapt to the specific needs of specialized domains.Table 6.1 summarizes the comparison between our framework (CSF) and Fedora, BRICKS and JeromeDL.

Compared with existing approaches, the main benefit of our approach lies in the high extensibility of each part of the system, the combination of key-

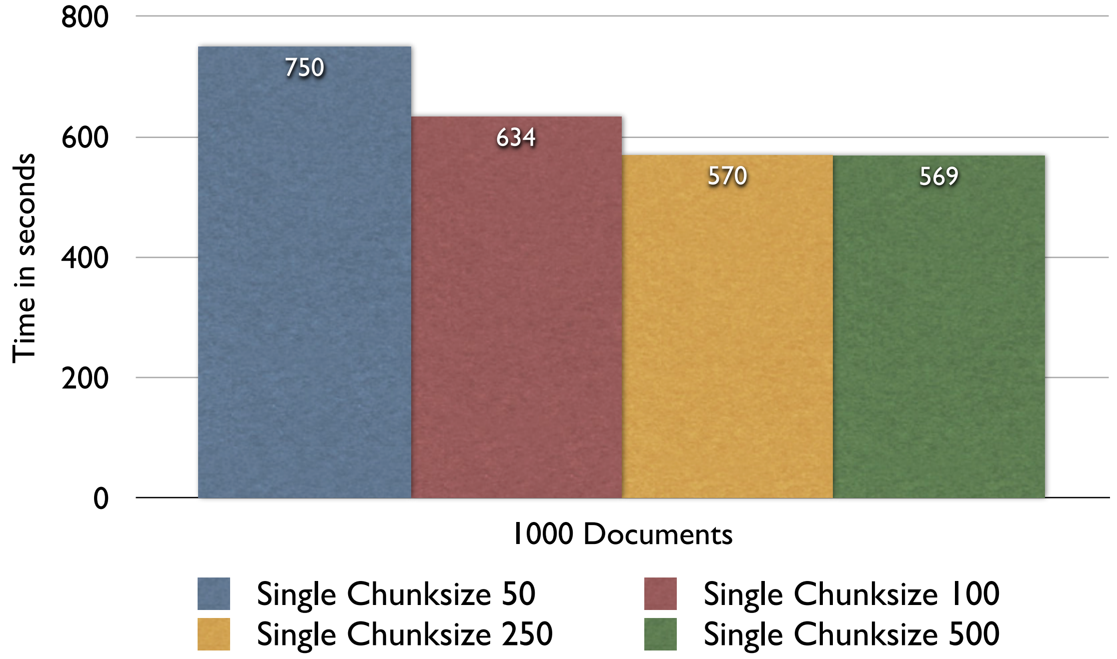


Figure 6.10: Keyword and Semantic Extraction Time in seconds for 1000 documents distributed

words, semantic and social search, as well as the possibility to incorporate any domain specific ontology. The component based approach guarantees that specific parts of the system can be easily exchanged with newer or more advanced implementations and, therefore, ensures a sustainable infrastructure for social semantic search. Furthermore, it allows the distribution of components which not only brings significant performance boosts, but also allows physical distribution of components over different sites. The simple API further enables integration of existing systems and therefore can support the different Legacy Infrastructures. The module necessary for the integration of said systems simply translates messages between these systems according to CSF's API.

In terms of interfaces our service based solution also enables the future integration of automated approaches, specifically it could be used for automatic interactions with the framework, automated search and retrieval as well as comparison. Another benefit of the service based approach is the easy extension to alternative platforms, examples would be mobile devices (like the iPhone or Android platform), as well as novel devices like the iPad. This enables the pervasive availability of our concept search framework as well as

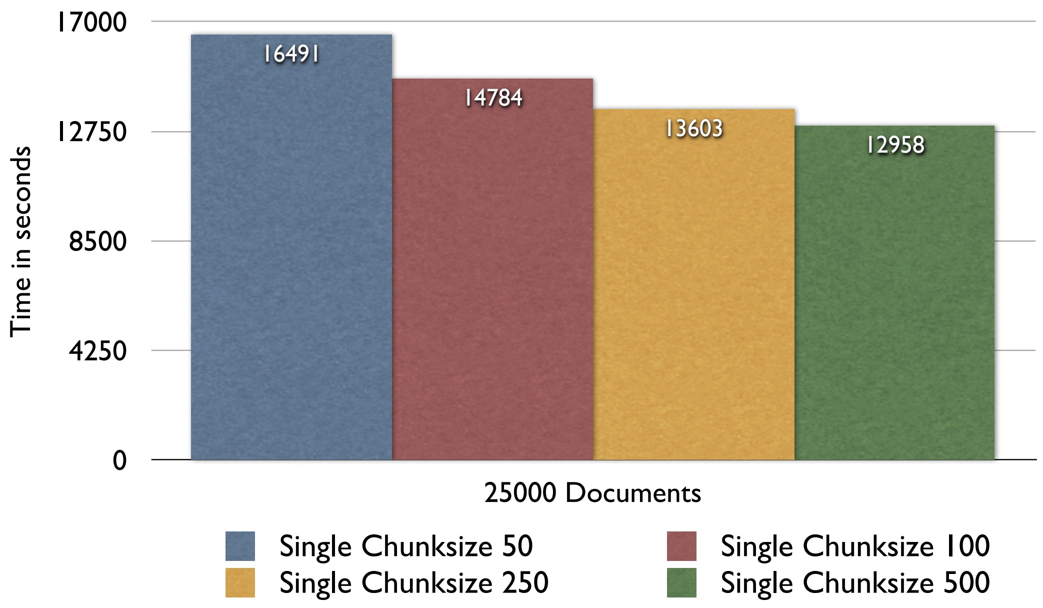


Figure 6.11: Keyword and Semantic Extraction Time in seconds for 25000 documents

novel and highly effective user experience enabled by these devices.

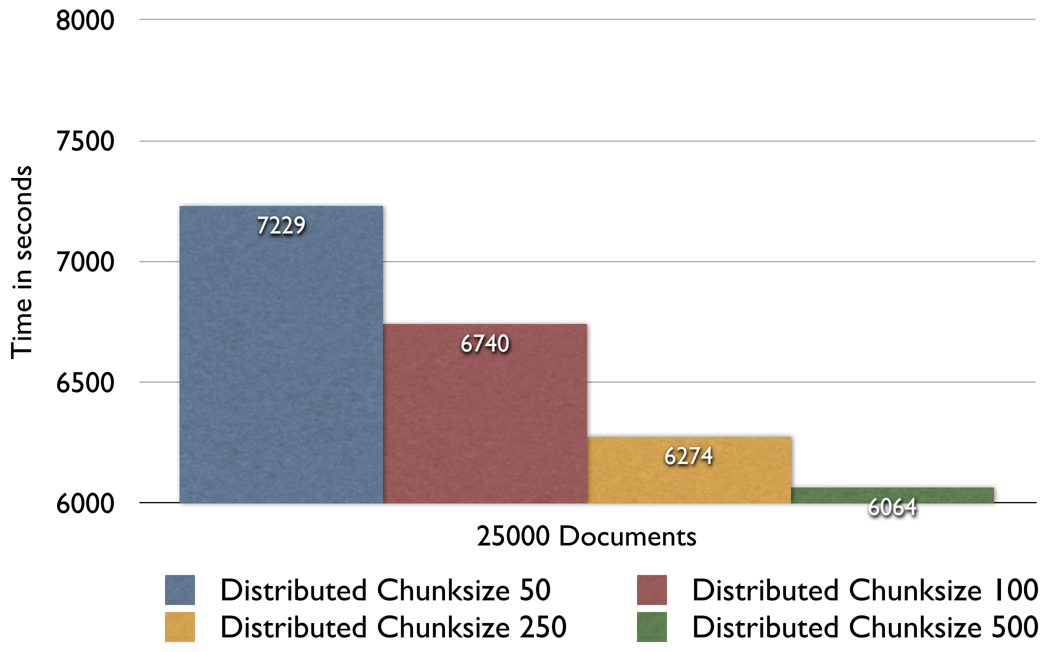


Figure 6.12: Keyword and Semantic Extraction Time in seconds for 25000 documents distributed

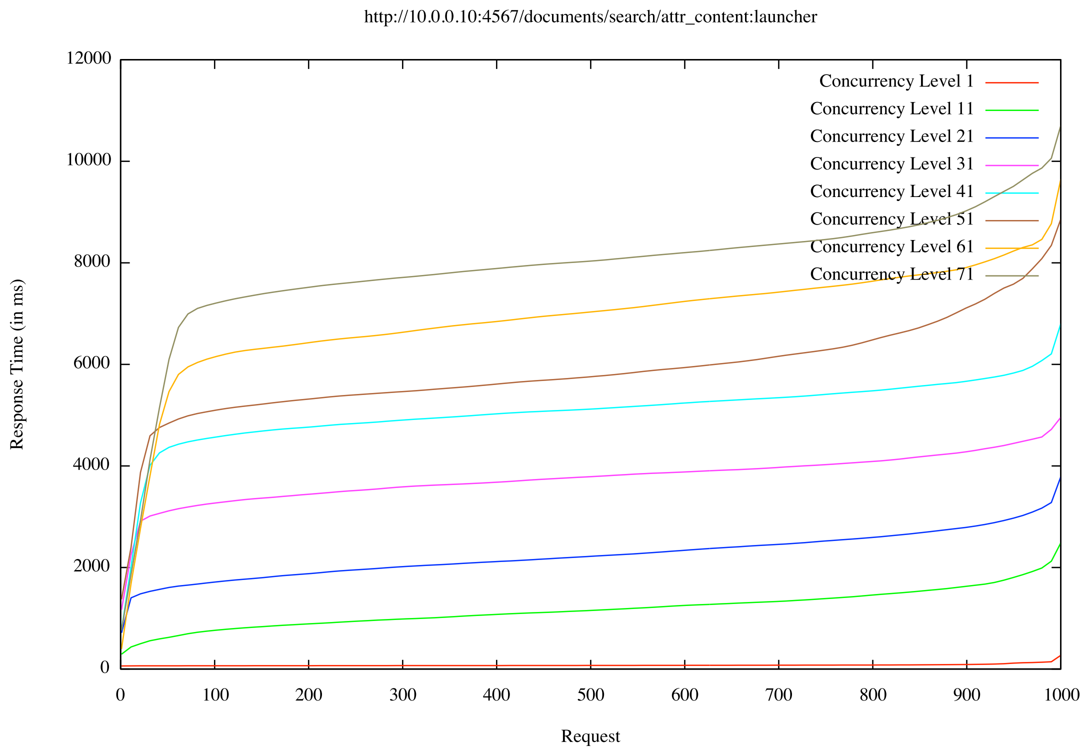


Figure 6.13: Keyword Query Performance

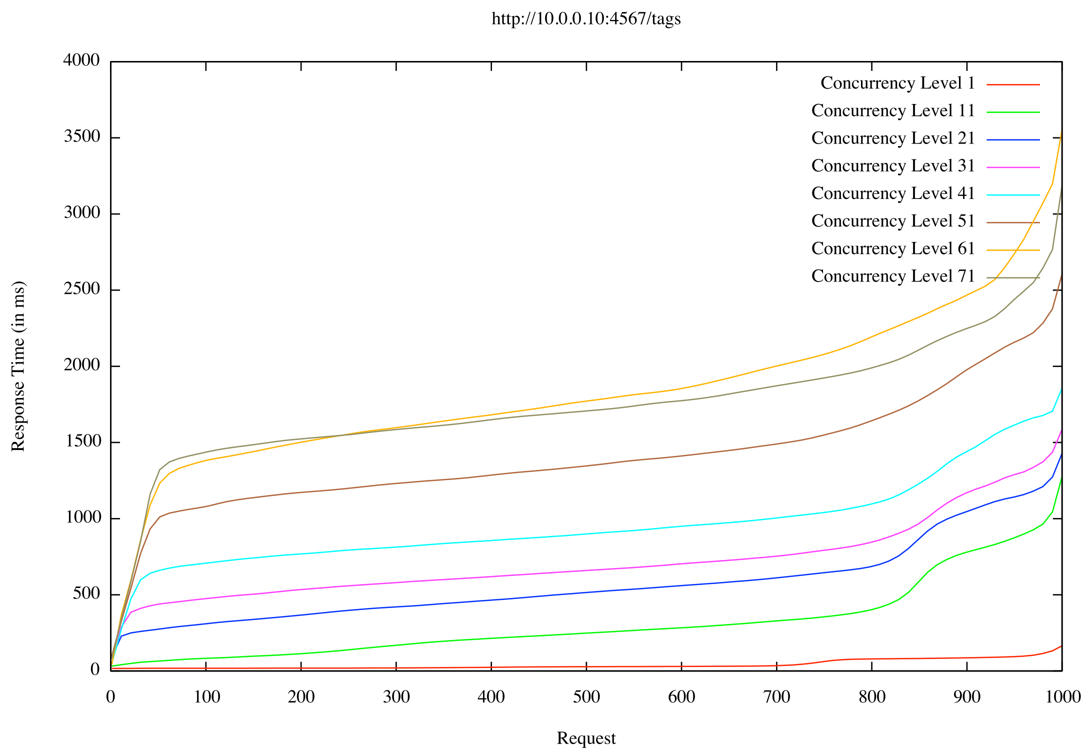


Figure 6.14: Concept Query Performance

Capabilities	**Fedora**	**BRICKS**	**JeromeDL**	**CSF**
Digital Object Support	Any	Any	Any	Any document in any format, images and videos if metadata is supplied
Semantic Extraction and Annotation	Middleware; External, output must comply with FOXML format and API	Middleware component	Supports metadata formats DublinCore, BibTex, MARC21	GATE, KIM, any legacy system or custom implementation that complies with the specified API
Storage Support	MySQL, Postgres, Oracle, McKoi; Kowari/-Mulgara	Any Jena-compliant backend	Any Sesame- or RDF2Go-compliant backend	Any (via Storage API)
Search Capabilities	Keyword, Semantic	Keyword, Semantic	Keyword, Semantic, Social	Keyword, Semantic, Social
User Participation	Tagging	Tagging	Knowledge Sharing via Blogs, Free Tagging, Wikis	Incremental Ontology Buildup, Game integration, Tagging
Interfaces	HTTP, REST, SOAP, FTP	Eclipse based rich client interface, Struts-based web interface, map visualizations of GIS-metadata	MultiBeeBrowse, TagsTreeMaps, Faceted Filtering	Visualizations 2D, 3D, Service Interface, Multiple Client interfaces.
Distribution and Parallelization Models	Federation via name resolver search services, Alvis P2P	Fully decentralized (P2P)	Distributed searching (P2P), aggregated browsing (hierarchical)	Any
Considerations of specific needs for the Specialized Domains	None	None	None	Incremental Ontology Buildup, Highly adaptable to user processes (e.g. CDF), Integration of any legacy system.

Table 6.1: Comparison of our proposed conceptual search framework (CSF) with existing digital library systems

7 Conclusion and Future Plans

7.1 Thesis results

In this thesis, we have presented a distributed social semantic search framework for specialized domains. We have discussed the state-of-the art on concept search and digital libraries, in general, as well as their specific support for specialized domains. Several tools and techniques have been examined. Based on the state-of-the art analysis we have identified typical use cases and a comprehensive list of requirements for the concept search framework.

Based on a critical analysis of use cases and requirements, a concept search framework for specialized domains has been developed. In this framework, we have analyzed and designed several components in an open flexible architecture so that the framework can be implemented by utilizing different existing techniques, on the one hand, and by incorporating new services and frameworks, on the other hand. We further developed a combined semantic representations that allows the integration of any Domain Specific Ontology and therefore builds a unique fundament for social semantic search in specialized domains. The proposed framework comes with a detailed feasibility study of implementation and a small-scale prototype of the framework to illustrate our concept search.

Techniques for concept search that combines semantic and social search in general have increasingly been researched and mature techniques are increasingly available. However, the exploitation of concept search for specialized domains has just been started. Therefore, any proposed concept search framework must be open enough to accommodate new developments in this domain. The main benefit of our approach lies in the high extendability of each part of the system. The component based approach guarantees that specific parts of the system can be easily exchanged to ensure a sustainable infrastructure for social semantic search. The simple API and service-based solution further enables integration of existing systems as well as the possibility to utilize only certain components of the system for specific tasks in specialized domains.

7.2 Future Work

In the case of *Digital Objects* current research as well as implementations are mostly concerned with text documents and their various formats. Images and Videos are reduced to the extractable or available metadata and by that means treated similar to text documents. Recent research, however, provides interesting new mechanism that could greatly change what is extractable and therefore searchable in terms of images and video. For images the most interesting new approach would be the broader adoption of compressive sensing technologies shown in [15] and [36]. These technologies would reduce the effective data size of images to an absolute minimum while being still providing an expressive and distinctive representation. This representation could be effectively used for search so that approaches where a user would identify certain objects in an image and similar images could be provided. In the case of video there is, on the one side, an emerging field of semantically enriched video storage approaches (seen in [102]), which use ontologies to enhance the expressiveness of video search solutions. On the other side, there is a possibility to utilize similarity search approaches to find videos as shown in [26]. The utilization of these new technologies would enable Images and Videos as first class citizens for search frameworks and greatly improve what can be searched and retrieved.

For *Semantic Extraction* the greatest room for future improvement lies in the area of user participation. The utilization of game based approaches on a large scale could greatly improve the accuracy of Semantic Extraction in general and is basically outlined in the works of [98]. Furthermore, the move from batch centered execution models like GATE to service based ones (as outlined in our architecture) would further enable cloud based Semantic Extraction Services that could find a widely larger adoption than current approaches, and therefore initiate the move to generally available semantic annotations that could be utilized. Another aspect of interest is the parallelization of the extraction process by using highly distributed processing approaches such as Map/Reduce

[32], enabling massively parallel execution of expensive extraction and annotation tasks [66], greatly improving performance. Another aspect is that in order to enable the accurate measurement of Precision and Recall for the space domain (which is currently not possible due to the lack of test data), a manually annotated test data set known as the Golden Standard should be generated by domain experts to accurately benchmark semantic extraction and search.

In terms of Storage the utilization of Document Oriented Storage solutions like CouchDB[1] (see Section 3.3.2) for Digital Object storage could greatly improve the capabilities of Search Frameworks. Being schema free any format of Digital Objects could be stored allowing greater dynamics for Digital Objects integration. Furthermore, the efficient data retrieval and horizontal scalability would greatly improve the overall performance of search and retrieval.

The area of Search and Indexing is undergoing constant improvement, and a variety of novel algorithms [see 35]) as well as the improvement of existing ones [see 81] will gradually improve the retrievable results. However, the most pressing current limitation lies not in search itself but in the available metadata to be searched, as well as the missing semantic annotations. This issue, however, will hopefully be addressed by the improvements taking place in the area of Semantic Extraction. Similar to Semantic Extraction, the area Search and Indexing can also greatly benefit from massively parallel computation using Map/Reduce, distributing the generation of indices, as well as the execution of index queries to multiple instances of replicated and shared indices to achieve maximum performance and scalability.

We conclude our recommendations with the discussion of future extension related to Interfaces and Visualizations. Despite the current approach of result presentation as ranked lists, several additional visualizations 2D as well as 3D have shown great potential, especially in the areas of concept and social search [85; 63; 28; 104].

[1] http://couchdb.apache.org/

A How to run the Concept Demonstration

In this section we describe the environment that needs to be established in order to run the Concept Demonstration.

The Concept Demonstration can be run under Unix / Linux / MacOS X 10.6.4, and the following software needs to be installed additionally:

Ruby Version 1.8.7p249

Java Version 6.0

Rubygems Version 1.3.7

Raptor RDF Parser toolkit Version 1.4.21

GATE Version 5.2.1 (`http://gate.ac.uk/download/`)

Solr Version 1.4.1 (`http://mirror.deri.at/apache/lucene/solr/1.4.1/`)

Sesame Version 2.3.2 (`http://sourceforge.net/projects/sesame/files/Sesame%202/`)

The following Rubygems need to be installed via `gem install`:

- `sinatra`
- `activesupport`
- `rdf`
- `rdf-raptor`

After installing Sesame, a new repository called `ESA`, using the template `native-rdfs-dt` is to be created. A detailed explanation on how to create the required repository can be found in [5].

Ensure that Sesame and Solr are started on their default ports. After that execute the following scripts

- `ruby importer.rb`
- `ruby extraction.rb`
- `ruby rest_server.rb`
- `ruby web_interface.rb`

You can now access the Web Interface via `http://localhost:3000`

B Experiment Sources

B.1 Data Model Ontology

```
<?xml version="1.0"?>


<!DOCTYPE rdf:RDF [
    <!ENTITY foaf "http://xmlns.com/foaf/0.1#" >
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY dc "http://purl.org/dc/elements/1.1/#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>


<rdf:RDF xmlns="http://www.blackwhale.at/ontologies/2010/06/↩
    common_sense.rdf#"
     xml:base="http://www.blackwhale.at/ontologies/2010/06/common_sense↩
         .rdf"
     xmlns:dc="http://purl.org/dc/elements/1.1/#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:foaf="http://xmlns.com/foaf/0.1#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <owl:Ontology rdf:about="http://www.blackwhale.at/ontologies↩
        /2010/06/common_sense.rdf"/>



    <!--
    ///////////////////////////
    //
    // Annotation properties
    //
    ///////////////////////////
     -->

    <owl:AnnotationProperty rdf:about="&rdfs;label"/>
```

```
<owl:AnnotationProperty rdf:about="&rdfs;comment"/>



<!--
//////////////////////////
//
// Object Properties
//
//////////////////////////
 -->




<!-- http://www.blackwhale.at/ontologies/2010/06/common_sense.rdf#↩
    author -->

<owl:ObjectProperty rdf:about="http://www.blackwhale.at/ontologies↩
    /2010/06/common_sense.rdf#author">
    <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
    <owl:equivalentProperty rdf:resource="&foaf;maker"/>
</owl:ObjectProperty>



<!-- http://www.blackwhale.at/ontologies/2010/06/common_sense.rdf#↩
    expertIn -->

<owl:ObjectProperty rdf:about="http://www.blackwhale.at/ontologies↩
    /2010/06/common_sense.rdf#expertIn">
    <rdfs:domain rdf:resource="http://www.blackwhale.at/ontologies↩
        /2010/06/common_sense.rdf#Person"/>
    <rdfs:range rdf:resource="http://www.blackwhale.at/ontologies↩
        /2010/06/common_sense.rdf#TopicExpertise"/>
    <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
</owl:ObjectProperty>



<!-- http://www.blackwhale.at/ontologies/2010/06/common_sense.rdf#↩
    expertiseTopic -->

<owl:ObjectProperty rdf:about="http://www.blackwhale.at/ontologies↩
    /2010/06/common_sense.rdf#expertiseTopic">
    <rdfs:domain rdf:resource="http://www.blackwhale.at/ontologies↩
        /2010/06/common_sense.rdf#TopicExpertise"/>
    <rdfs:range rdf:resource="&owl;Thing"/>
    <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
</owl:ObjectProperty>
```

```
<!-- http://www.blackwhale.at/ontologies/2010/06/common_sense.rdf#↩
    hasExpertiseFactor -->

<owl:ObjectProperty rdf:about="http://www.blackwhale.at/ontologies↩
    /2010/06/common_sense.rdf#hasExpertiseFactor">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:range rdf:resource="http://www.blackwhale.at/ontologies↩
        /2010/06/common_sense.rdf#ExpertiseFactor"/>
    <rdfs:domain rdf:resource="http://www.blackwhale.at/ontologies↩
        /2010/06/common_sense.rdf#TopicExpertise"/>
    <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
</owl:ObjectProperty>



<!-- http://www.blackwhale.at/ontologies/2010/06/common_sense.rdf#↩
    hasInterestFactor -->

<owl:ObjectProperty rdf:about="http://www.blackwhale.at/ontologies↩
    /2010/06/common_sense.rdf#hasInterestFactor">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:range rdf:resource="http://www.blackwhale.at/ontologies↩
        /2010/06/common_sense.rdf#InterestFactor"/>
    <rdfs:domain rdf:resource="http://www.blackwhale.at/ontologies↩
        /2010/06/common_sense.rdf#TopicInterest"/>
    <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
</owl:ObjectProperty>



<!-- http://www.blackwhale.at/ontologies/2010/06/common_sense.rdf#↩
    hasInterestIn -->

<owl:ObjectProperty rdf:about="http://www.blackwhale.at/ontologies↩
    /2010/06/common_sense.rdf#hasInterestIn">
    <rdfs:domain rdf:resource="http://www.blackwhale.at/ontologies↩
        /2010/06/common_sense.rdf#Person"/>
    <rdfs:range rdf:resource="http://www.blackwhale.at/ontologies↩
        /2010/06/common_sense.rdf#TopicInterest"/>
    <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
</owl:ObjectProperty>



<!-- http://www.blackwhale.at/ontologies/2010/06/common_sense.rdf#↩
    topic -->

<owl:ObjectProperty rdf:about="http://www.blackwhale.at/ontologies↩
    /2010/06/common_sense.rdf#topic">
```

```
        <rdfs:domain rdf:resource="http://www.blackwhale.at/ontologies↩
            /2010/06/common_sense.rdf#TopicInterest"/>
        <rdfs:range rdf:resource="&owl;Thing"/>
        <rdfs:subPropertyOf rdf:resource="&owl;topObjectProperty"/>
    </owl:ObjectProperty>



    <!-- http://www.w3.org/2002/07/owl#topObjectProperty -->

    <owl:ObjectProperty rdf:about="&owl;topObjectProperty"/>



    <!-- http://xmlns.com/foaf/0.1#maker -->

    <owl:ObjectProperty rdf:about="&foaf;maker"/>



    <!--
    ///////////////////////////
    //
    // Data properties
    //
    ///////////////////////////
     -->




    <!-- http://purl.org/dc/elements/1.1/#title -->

    <owl:DatatypeProperty rdf:about="&dc;title"/>



    <!-- http://www.blackwhale.at/ontologies/2010/06/common_sense.rdf#↩
        expertiseFactor -->

    <owl:DatatypeProperty rdf:about="http://www.blackwhale.at/↩
        ontologies/2010/06/common_sense.rdf#expertiseFactor">
        <rdfs:domain rdf:resource="http://www.blackwhale.at/ontologies↩
            /2010/06/common_sense.rdf#ExpertiseFactor"/>
        <rdfs:range rdf:resource="&xsd;integer"/>
        <rdfs:subPropertyOf rdf:resource="&owl;topDataProperty"/>
    </owl:DatatypeProperty>
```

```
<!-- http://www.blackwhale.at/ontologies/2010/06/common_sense.rdf#↩
    interestFactor -->

<owl:DatatypeProperty rdf:about="http://www.blackwhale.at/↩
    ontologies/2010/06/common_sense.rdf#interestFactor">
    <rdfs:domain rdf:resource="http://www.blackwhale.at/ontologies↩
        /2010/06/common_sense.rdf#InterestFactor"/>
    <rdfs:range rdf:resource="&xsd;integer"/>
    <rdfs:subPropertyOf rdf:resource="&owl;topDataProperty"/>
</owl:DatatypeProperty>



<!-- http://www.w3.org/2002/07/owl#topDataProperty -->

<owl:DatatypeProperty rdf:about="&owl;topDataProperty"/>



<!--
///////////////////////////
//
// Classes
//
///////////////////////////
 -->




<!-- http://www.blackwhale.at/ontologies/2010/06/common_sense.rdf#↩
    DigitalObject -->

<owl:Class rdf:about="http://www.blackwhale.at/ontologies/2010/06/↩
    common_sense.rdf#DigitalObject">
    <rdfs:label xml:lang="en">digital object</rdfs:label>
    <rdfs:subClassOf rdf:resource="&foaf;Document"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&dc;title"/>
            <owl:minCardinality rdf:datatype="&xsd;↩
                nonNegativeInteger">1</owl:minCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&foaf;maker"/>
            <owl:minCardinality rdf:datatype="&xsd;↩
                nonNegativeInteger">1</owl:minCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
```

```
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="&foaf;maker"/>
                <owl:allValuesFrom rdf:resource="http://www.blackwhale.↩
                    at/ontologies/2010/06/common_sense.rdf#Person"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:comment xml:lang="en">The basic unit of information.</↩
            rdfs:comment>
    </owl:Class>



    <!-- http://www.blackwhale.at/ontologies/2010/06/common_sense.rdf#↩
        ExpertiseFactor -->

    <owl:Class rdf:about="http://www.blackwhale.at/ontologies/2010/06/↩
        common_sense.rdf#ExpertiseFactor"/>



    <!-- http://www.blackwhale.at/ontologies/2010/06/common_sense.rdf#↩
        Group -->

    <owl:Class rdf:about="http://www.blackwhale.at/ontologies/2010/06/↩
        common_sense.rdf#Group">
        <owl:equivalentClass rdf:resource="&foaf;Group"/>
        <rdfs:comment xml:lang="en">The Group provides an ↩
            organizational mechanism to aggregate multiple Persons in a ↩
            Team</rdfs:comment>
    </owl:Class>



    <!-- http://www.blackwhale.at/ontologies/2010/06/common_sense.rdf#↩
        InterestFactor -->

    <owl:Class rdf:about="http://www.blackwhale.at/ontologies/2010/06/↩
        common_sense.rdf#InterestFactor"/>



    <!-- http://www.blackwhale.at/ontologies/2010/06/common_sense.rdf#↩
        Person -->

    <owl:Class rdf:about="http://www.blackwhale.at/ontologies/2010/06/↩
        common_sense.rdf#Person">
        <owl:equivalentClass rdf:resource="&foaf;Person"/>
        <rdfs:comment xml:lang="en">A person is either directly ↩
            involved in the creation of digital objects, or, is only ↩
            associated to DOs due to existing interests.</rdfs:comment>
    </owl:Class>



    <!-- http://www.blackwhale.at/ontologies/2010/06/common_sense.rdf#↩
        TopicExpertise -->

    <owl:Class rdf:about="http://www.blackwhale.at/ontologies/2010/06/↩
        common_sense.rdf#TopicExpertise">
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="http://www.blackwhale.at/↩
                    ontologies/2010/06/common_sense.rdf#expertiseFactor"↩
                    />
                <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger"↩
                    >1</owl:cardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://www.blackwhale.at/ontologies/2010/06/common_sense.rdf#↩
        TopicInterest -->

    <owl:Class rdf:about="http://www.blackwhale.at/ontologies/2010/06/↩
        common_sense.rdf#TopicInterest">
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="http://www.blackwhale.at/↩
                    ontologies/2010/06/common_sense.rdf#interestFactor"/↩
                    >
                <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger"↩
                    >1</owl:cardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://www.w3.org/2002/07/owl#Thing -->

    <owl:Class rdf:about="&owl;Thing"/>



    <!-- http://xmlns.com/foaf/0.1#Document -->

    <owl:Class rdf:about="&foaf;Document"/>

    <!-- http://xmlns.com/foaf/0.1#Group -->

    <owl:Class rdf:about="&foaf;Group"/>



    <!-- http://xmlns.com/foaf/0.1#Person -->

    <owl:Class rdf:about="&foaf;Person"/>
</rdf:RDF>
```

Listing B.1: Automatic Gazetteer List Generator

B.2 Automatic Gazetteer List Generator

```
#!/usr/bin/env ruby -rubygems

begin
  require 'rdf'
  require 'rdf/raptor'
  require 'active_support'
rescue LoadError
  puts "One or more of the required gems could not be found!"
  puts "Install required gems using\n\tgem install rdf rdf-raptor ↩
      active_support"
  puts "Also, please make sure to install the 'raptor' RDF parser."
  exit
end

dirname = File.join(File.dirname(__FILE__), "..", "SWEET")
path = File.join(dirname, "*.owl")

$output_dir = output_dir = File.join(File.dirname(__FILE__), "↩
    gazetteer_files")
Dir.mkdir(output_dir) unless File.exists?(output_dir)

def find_subclasses_of(graph, class_uri)
  subclasses = []
  graph.query([nil, RDF::RDFS.subClassOf, class_uri]) do |stmt|
    print "."; STDOUT.flush
    subclasses << stmt.subject
    subclasses << find_instances_of(graph, stmt.subject)
    subclasses += find_subclasses_of(graph, stmt.subject)
  end
  subclasses
end
```

```
def find_all_subclasses_of(graphs, class_uri)
  graphs.collect do |graph|
    find_subclasses_of(graph, class_uri)
  end.flatten.compact
end

def find_instances_of(graph, class_uri)
  graph.query([nil, RDF.type, class_uri]).collect(&:subject)
end


def load_graphs(path, format = :rdfxml)
  Dir[path].collect do |file|
    RDF::Graph.load(file, :format => format)
  end.compact
end


def fill_list_for(concept, output_dir = $output_dir)
  concept_name = concept.to_s.split("#").last
  list_file_name = concept_name.underscore + "_list"

  # For demonstration purposes, add only a humanized version of the ↩
      concept
  # name to the list file.
  humanized_name = concept_name.underscore.humanize

  puts "Will␣add␣'#{humanized_name}'␣to␣#{list_file_name}"
  File.open(File.join(output_dir, list_file_name), "w") do |file|
    file.puts humanized_name
  end

  list_file_name
end

def write_mapping_definitions(mapping_definitions, output_dir = ↩
    $output_dir)
  File.open(File.join(output_dir, "mapping_definitions.def"), "w") do |↩
      file|
    file.puts mapping_definitions.join("\n")
  end
end

# Physical phenomena url:
# http://sweet.jpl.nasa.gov/2.0/phys.owl#PhysicalPhenomena
sweet_base_uri = "http://sweet.jpl.nasa.gov/2.0"
base_classes = [
  RDF::URI.new("#{sweet_base_uri}/phys.owl#PhysicalPhenomena"),
  RDF::URI.new("#{sweet_base_uri}/astroPlanet.owl#PlanetaryRealm")
]
```

```
puts "Loading␣Ontology..."
graphs = load_graphs(path)

subclasses = base_classes.collect do |base_class|
  puts "Finding␣all␣subclasses␣of␣#{base_class}..."
  find_all_subclasses_of(graphs, base_class)
end.flatten.compact
puts

puts "Generating␣gazetteer␣list␣for␣each␣identified␣concept..."
mapping_definitions = []
subclasses.each do |concept|
  # Fill Gazetteer List
  list_file_name = fill_list_for(concept)

  # Entry in the mapping definition:
  # <list_file_name>:<concept_uri>
  mapping_definitions << "#{list_file_name}:#{concept.to_s.gsub("#", ↩
      ":")}"
end

puts "Writing␣Mapping␣definitons..."
write_mapping_definitions(mapping_definitions)
```

Listing B.2: Automatic Gazetteer List Generator

B.3 GATE Batch Process Wrapper

```
/*
 * Gate Document Batch Processor
 *
 * Adapted from http://doiop.com/GateBatchProcessApp.java
 */
import gate.Document;
import gate.Corpus;
import gate.CorpusController;
import gate.AnnotationSet;
import gate.Annotation;
import gate.Gate;
import gate.Factory;
import gate.util.*;
import gate.util.persistence.PersistenceManager;


import org.openrdf.OpenRDFException;
import org.openrdf.model.URI;
import org.openrdf.model.ValueFactory;
import org.openrdf.model.vocabulary.RDF;
import org.openrdf.model.vocabulary.RDFS;
import org.openrdf.repository.Repository;
import org.openrdf.repository.RepositoryConnection;
import org.openrdf.repository.http.HTTPRepository;


import java.util.Set;
import java.util.HashSet;
import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;

import java.io.File;
import java.io.FileOutputStream;
import java.io.BufferedOutputStream;
import java.io.OutputStreamWriter;
import java.io.FileReader;
import java.io.BufferedReader;


/**
 * This class loads a GATE application and runs it against the provided↩
     files.
 */
public class BatchProcessApp {
  public static void main(String[] args) throws Exception {
    parseCommandLine(args);

    // initialise GATE - this must be done before calling any GATE APIs
    Gate.init();

    Repository myRepository = new HTTPRepository(sesameServer, ↩
        repositoryID);
    myRepository.initialize();

    ValueFactory f = myRepository.getValueFactory();


    // load the saved application
    CorpusController application =
      (CorpusController)PersistenceManager.loadObjectFromFile(gappFile)↩
        ;

    Corpus corpus = Factory.newCorpus("BatchProcessApp␣Corpus");
    application.setCorpus(corpus);

    for(int i = firstFile; i < args.length; i++) {
      File idFile = new File(args[i]);
      File docFile = new File(workingDir + "/" +
                        new BufferedReader(new FileReader(idFile)).↩
                            readLine());
      System.out.print("Processing␣document␣" + docFile + "...");
      Document doc = null;
      try {
        doc = Factory.newDocument(docFile.toURI().toURL(), encoding);
      } catch(Exception e) {
        System.out.println("FAILED.");
        continue;
      }

      // Add the document to the corpus, run the application, and ↩
          finally
      // remove the document from the corpus to prepare it for the next↩
           run.
      corpus.add(doc);
      application.execute();
      corpus.clear();

      String docXMLString = null;

      System.out.println("Annotations:␣" + doc.getAnnotations("onto").↩
          toString());

      // Store all generated Annotations in the Sesame RDF store.
      for(Annotation a:doc.getAnnotations("onto")) {
        URI concept = f.createURI(a.getFeatures().get("ontology")+"#"+a↩
            .getFeatures().get("class"));
        URI document = f.createURI(docFile.getCanonicalFile().toURI().↩
            toURL().toString());
        URI topic = f.createURI("http://xmlns.com/foaf/0.1/primaryTopic↩
            ");

        try {
          RepositoryConnection con = myRepository.getConnection();

          try {
            con.add(document, topic, concept);
          } finally {
            con.close();
          }
        } catch (OpenRDFException e) {
            // handle exception
        }
      }

      Factory.deleteResource(doc);

      System.out.println("done");
    }

    System.out.println("All␣done");
  }
```

```
private static void parseCommandLine(String[] args) throws Exception ↩
    {
  System.out.println("Processing:␣" + args.toString());
  int i;
  for(i = 0; i < args.length && args[i].charAt(0) == '-'; i++) {
    switch(args[i].charAt(1)) {
      case 'g':
        gappFile = new File(args[++i]);
        break;
      case 's':
        sesameServer = args[++i];
        System.out.println("Sesame␣Server:␣" + sesameServer);
        break;
      case 'r':
        repositoryID = args[++i];
        System.out.println("Repository␣ID:␣" + repositoryID);
        break;
      case 'w':
        workingDir = args[++i];
        System.out.println("Working␣Dir:␣" + workingDir);
        break;
      default:
        System.out.println("Unrecognized␣option␣" + args[i]);
        usage();
    }
  }

  firstFile = i;

  if(gappFile == null) {
    System.err.println("No␣.gapp␣file␣specified");
    usage();
  }
}

private static final void usage() {
  System.err.println(
    "Usage:\n" +
    "␣␣␣java␣BatchProcessApp␣-g␣<gappFile>␣file1␣file2␣...␣fileN\n" +
    "\n" +
    "-g␣gappFile\n" +
    "-r␣repositoryId\n" +
    "-s␣sesameServer\n" +
    "-w␣workingDir\n"
 );

  System.exit(1);
}
```

```
    private static int firstFile = 0;
    private static File gappFile = null;
    private static String encoding = null;
    // Sesame configuration
    private static String sesameServer = "http://localhost:8080/openrdf-↩
        sesame/";
    private static String repositoryID = "ESA";
    private static String workingDir = "/tmp/esa";
}
```

Listing B.3: GATE Batch Process Wrapper

B.4 GATE Application Skeleton for Semantic Extraction

```
<gate.util.persistence.GateApplication>
  <application class="gate.util.persistence.↩
      SerialAnalyserControllerPersistence">
    <prList class="gate.util.persistence.CollectionPersistence">
      <localList>
        <gate.util.persistence.LanguageAnalyserPersistence>
          <runtimeParams class="gate.util.persistence.MapPersistence">
            <mapType>gate.util.SimpleFeatureMapImpl</mapType>
            <localMap>
              <entry>
                <string>annotationSetName</string>
                <string>onto</string>
              </entry>
              <entry>
                <string>document</string>
                <null/>
              </entry>
            </localMap>
          </runtimeParams>
          <resourceType>gate.creole.gazetteer.OntoGazetteerImpl</↩
              resourceType>
          <resourceName>OntoGazetteer_000A8</resourceName>
          <initParams class="gate.util.persistence.MapPersistence">
            <mapType>gate.util.SimpleFeatureMapImpl</mapType>
            <localMap>
              <entry>
                <string>listsURL</string>
                <gate.util.persistence.PersistenceManager-URLHolder>
                  <urlString>$relpath$gaz_definitions/↩
                      mapping_definitions.def</urlString>
                </gate.util.persistence.PersistenceManager-URLHolder>
              </entry>
              <entry>
                <string>encoding</string>
                <string>UTF-8</string>
              </entry>
              <entry>
                <string>caseSensitive</string>
                <boolean>true</boolean>
              </entry>
              <entry>
                <string>gazetteerName</string>
                <string>com.ontotext.gate.gazetteer.HashGazetteer</↩
                    string>
              </entry>
              <entry>
                <string>mappingURL</string>
                <gate.util.persistence.PersistenceManager-URLHolder>
                  <urlString>$relpath$gaz_definitions/↩
                      mapping_definitions.def</urlString>
                </gate.util.persistence.PersistenceManager-URLHolder>
              </entry>
            </localMap>
          </initParams>
          <features class="gate.util.persistence.MapPersistence">
            <mapType>gate.util.SimpleFeatureMapImpl</mapType>
            <localMap/>
          </features>
        </gate.util.persistence.LanguageAnalyserPersistence>
      </localList>
      <collectionType>java.util.ArrayList</collectionType>
    </prList>
    <resourceType>gate.creole.SerialAnalyserController</resourceType>
    <resourceName>ANNIE</resourceName>
    <initParams class="gate.util.persistence.MapPersistence">
      <mapType>gate.util.SimpleFeatureMapImpl</mapType>
      <localMap/>
    </initParams>
    <features class="gate.util.persistence.MapPersistence">
      <mapType>gate.util.SimpleFeatureMapImpl</mapType>
      <localMap/>
    </features>
  </application>
</gate.util.persistence.GateApplication>
```

Listing B.4: GATE Application Skeleton for Semantic Extraction

B.5 ESA Document Importer

```
# require 'appscript'
require 'nokogiri'
require 'open-uri'
```

```

module Commonsense
  module Importer
    class EsaImport
      def self.import


        plainUrl = "http://www.esa.int/SPECIALS/GSP/"
        download = []
        [
          "http://www.esa.int/SPECIALS/GSP/SEMBHQYO4HD_0.html",
          "http://www.esa.int/SPECIALS/GSP/SEMJIQYO4HD_0.html",
          "http://www.esa.int/SPECIALS/GSP/SEMMHQYO4HD_0.html"
        ].each do |url|
          doc = Nokogiri::HTML(open(url))

          # Get all links


          links = doc.xpath("//a")
          links.each do |link|
            if link.content =~ /^\d{2,}\/.{3,}/
              #puts "\n#{link.content}: #{link['href']}"
              summary = Nokogiri::HTML(open("#{plainUrl}#{link['href']}↩
                  "))
              summary_links = summary.xpath("//a")
              summary_links.each do |slink|

                if slink.content == "Executive␣summary"
                  #puts "\n#{slink.content}: #{slink['href']}"
                  download << slink['href']
                end
              end
            end
          end

        end

        download.each do |download|
          system("cd␣#{File.dirname(__FILE__)}/esa_corpus␣&&␣curl␣-O␣↩
              '#{download}'")

        end

      end
    end
  end
end
```

Listing B.5: ESA Document Importer

B.6 ESA Load Simulator

```
module Commonsense
  module Importer
    class EsaLoadSimulator
      def self.import

        source_dir = File.join(File.dirname(__FILE__), "esa_corpus")
        $destination_dir = File.join(File.dirname(__FILE__), "↩
            esa_load_corpus")
        source_path = File.join(source_dir, "*.pdf")

        duplication_factor = 100

        Dir[source_path].each do |file|
          (1..duplication_factor).each do |i|
            write_mapping_file(i.to_s+File.basename(file), file)
          end
        end

      end

      def self.write_mapping_file(mapped_file_name, file_name, ↩
          destination_dir = $destination_dir)
        File.open(File.join(destination_dir, "#{mapped_file_name}.txt")↩
            , "w") do |file|
          file.puts file_name
        end
      end

    end
  end
end
```

Listing B.6: ESA Load Simulator

B.7 Rest Server / Search Module

```
require 'sinatra'
require File.join(File.dirname(__FILE__), 'commonsense')

include Commonsense::Core

# class RestServer < Sinatra::Application

get %r{^/(?:stats)?(?:\.([\w]+))?$} do |format|
  format ||= "json"
  {
    "id" => "common_sense", "version" => "1",
    "stats" => {
      "documents" => Document.count,
      "tags" => Tag.count,
      "users" => User.count
    }
  }.send("to_#{format}")
end

get %r{^/documents(?:\.([\w]+))?$} do |format|
  format ||= "json"
  Document.all.send("to_#{format}", :methods => [:summary, :doc_type])
end

post %r{^/documents(?:\.([\w]+))?$} do |format|
  format ||= "json"
  d = Document.new(
    :name => params[:name],
    :document => params[:document],
    :created_at => (DateTime.parse(params[:created_at]) rescue nil)
  )
  if d.save
    "ok"
  else
    puts d.errors.full_messages.inspect
    halt 409, d.errors.to_json
  end
end

get '/documents/search/:query' do
  uri = Commonsense::Config[:keyword_index]

  response = eval(RestClient.get(uri+"?wt=ruby&fl=id+title+score+↩
      content_type&hl=true&hl.fl=attr_content&q=#{CGI.escape␣params[:↩
      query]}").body)
  documents = response["response"]["docs"].map do |doc|
    doc.merge({
      :snippets => response["highlighting"][doc["id"]].collect { |k,v| ↩
          v }.flatten,
      :summary => response["highlighting"][doc["id"]].collect { |k,v| v↩
          }.join("␣...␣"),
      :doc_type => doc["content_type"].first,
      :name => doc["title"] && doc["title"].first || doc["id"],
      :uri => doc["id"]
    })
  end
  documents.to_json
end

get '/documents/:id/temporally_related' do
  d = Document.find(params[:id])
```

```ruby
    d ||= Document.find_by_uri(params[:id])
    params[:dt] ||= 5
    dt = params[:dt].to_i.days
    from = d.created_at - dt
    to = d.created_at + dt
    options = { :conditions => { :created_at => from..to } }
    dt_documents = Document.all(options.merge(:limit => 50))
    dt_documents.to_json(:methods => [:summary, :doc_type])
  end

  get %r{^/documents/by_name(?:\.([\w]+))?} do |format|
    format ||= "json"
    Document.all(
      :conditions => [ 'name␣LIKE␣?', "%#{params[:name]}%" ],
      :limit => 25
    ).to_json(:methods => [ :summary, :doc_type ])
  end

  get %r{^/documents/by_uri(?:\.([\w]+))?} do |format|
    format ||= "json"
    Document.all(
      :conditions => [ 'uri␣LIKE␣?', "%#{params[:uri]}%" ],
      :limit => 25
    ).to_json(:methods => [ :summary, :doc_type ])
  end

  get '/documents/:id' do
    d = Document.find(params[:id])
    d ||= Document.find_by_uri(params[:id])
    methods = [:summary, :doc_type]
    methods << :document if params[:include_doc] == "true"
    JSON.parse(d.to_json(
      :methods => methods,
      :include => {
        :outgoing_document_relations => {
          :include => [:destination, :ratings]
        },
        :ratings => {}
      }
    )).merge!(
      :related_documents => d.relations,
      :tags => d.tags
    ).to_json
  end

  post %r{^/tags(?:\.([\w]+))?$} do |format|
    if t = Tag.create(:name => params[:name])
      "ok"
    else
      halt 405, t.errors.to_json
    end
```

```
end

get '/tags/by_name/:name' do
  Tag.all(:conditions => ['name␣LIKE␣?', "%#{params[:name]}%"]).to_json
end

get '/tags/documents' do
  uri = Commonsense::Config[:rdf_store]

  query = %Q{
␣␣␣␣PREFIX␣foaf:<http://xmlns.com/foaf/0.1/>
␣␣␣␣PREFIX␣rdfs:<http://www.w3.org/2000/01/rdf-schema#>

␣␣␣␣SELECT␣DISTINCT␣?document␣WHERE␣{
␣␣␣␣␣␣?document␣foaf:primaryTopic␣?topic␣.
␣␣␣␣␣␣?topic␣rdfs:subClassOf␣<#{params[:id]}>
    }
  }

  request = RestClient.get(uri+"?query="+CGI.escape(query), :accept => ↩
      "application/sparql-results+json")
  response = JSON.parse(request.body)
  puts response.inspect
  response["results"]["bindings"].collect { |b|
    { :id => b["document"]["value"] }
  }.to_json

end

get %r{^/tags(?:\.([\w]+))?} do |format|
  uri = Commonsense::Config[:rdf_store]

  query = %q{
␣␣␣␣PREFIX␣rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
␣␣␣␣PREFIX␣rdfs:<http://www.w3.org/2000/01/rdf-schema#>
␣␣␣␣PREFIX␣phys:<http://sweet.jpl.nasa.gov/2.0/phys.owl#>
␣␣␣␣PREFIX␣foaf:<http://xmlns.com/foaf/0.1/>
␣␣␣␣PREFIX␣owl:<http://www.w3.org/2002/07/owl#>
␣␣␣␣PREFIX␣astroPlanet:<http://sweet.jpl.nasa.gov/2.0/astroPlanet.owl#>
␣␣␣␣PREFIX␣sesame:<http://www.openrdf.org/schema/sesame#>

␣␣␣␣SELECT␣DISTINCT␣?class␣?directsuperclass␣WHERE␣{
␣␣␣␣␣␣?class␣sesame:directSubClassOf␣?directsuperclass␣.
␣␣␣␣␣␣{␣?directsuperclass␣rdfs:subClassOf␣astroPlanet:PlanetaryRealm␣} ↩
    UNION
      { ?directsuperclass rdfs:subClassOf phys:PhysicalPhenomena }
    }
  }

  request = RestClient.get(uri+"?query="+CGI.escape(query), :accept => ↩
      "application/sparql-results+json")
    response = JSON.parse(request.body)
    puts response.inspect
    response["results"]["bindings"].collect { |b|
      { :name => b["class"]["value"], :superclass => b["directsuperclass"]↩
          ]["value"] }
    }.to_json
  end


  get %r{^/document_relations(?:\.([\w]+))?$} do |format|
    format ||= "json"
    DocumentRelation.all.send("to_#{format}")
  end

  post %r{^/document_relations(?:\.([\w]+))?$} do |format|
    format ||= "json"
    begin
      DocumentRelation.create_undirected(Document.find(params[:one]), ↩
          Document.find(params[:two]))
      "ok"
    rescue
      halt 409, $!.inspect
    end
  end

  get %r{^/tag_document_relations(?:\.([\w]+))?$} do |format|
    format ||= "json"
    TagDocumentRelation.all.send("to_#{format}")
  end

  post %r{^/tag_document_relations(?:\.([\w]+))?$} do |format|
    format ||= "json"
    if r = TagDocumentRelation.create(:tag => Tag.find(params[:tag]), :↩
        document => Document.find(params[:document]))
      "ok"
    else
      halt 409, r.errors.send("to_#{format}")
    end
  end

  get '/tag_relations' do
    TagRelation.all.to_json
  end

  post '/tag_relations' do
    if r = TagRelation.create_undirected(Tag.find(params[:one]), Tag.find↩
        (params[:two]))
      "ok"
    else
      halt 409, r.errors.to_json
    end
```

```
end

get %r{^/ratings(?:\.([\w]+))?$} do |format|
  format ||= "json"
  Rating.all.send("to_#{format}")
end

post %r{^/ratings(?:\.([\w]+))?$} do |format|
  format ||= "json"
  r = Rating.new(:name => params[:name], :value => params[:value], :↩
      rateable_id => params[:rateable_id], :rateable_type => params[:↩
      rateable_type])
  if r.save
    "ok"
  else
    halt 409, r.errors.send("to_#{format}")
  end
end

get %r{^/users(?:\.([\w]+))?$} do |format|
  format ||= "json"
  User.all.send("to_#{format}")
end

get %r{^/users/by_name(?:\.([\w]+))?$} do |format|
  format ||= "json"
  name = params[:name].split
  conditions = { :first_name => name.first , :last_name => name.last }
  User.all(:conditions => conditions).send("to_#{format}")
end

get %r{^/users/(\d+)(?:\.([\w]+))?$} do |user_id, format|
  format ||= "json"
  User.find(user_id).send("to_#{format}")
end

get %r{^/users/(\d+)/documents(?:\.([\w]+))?$} do |user_id, format|
  format ||= "json"
  User.find(user_id).documents.send("to_#{format}", :methods => [:↩
      summary, :doc_type])
end


get '/search/time' do
  from = Date.parse(params[:from])
  to = Date.parse(params[:to])
  options = { :conditions => { :created_at => from..to } }
  {
    :users => User.all(options),
    :documents => Document.all(options),
    :tags => Tag.all(options),
      :document_relations => DocumentRelation.all,
      :tag_relations => TagRelation.all,
      :tag_document_relations => TagDocumentRelation.all
    }
  end


  get '/graphs/documents' do
    {
      :documents => Document.all,
      :relations => DocumentRelation.all
    }.to_json
  end

  get '/graphs/tags' do
    {
      :tags => Tag.all,
      :relations => TagRelation.all
    }.to_json
  end

  get '/graphs/full' do
    {
      :users => User.all,
      :documents => Document.all,
      :tags => Tag.all,
      :document_relations => DocumentRelation.all,
      :tag_relations => TagRelation.all,
      :tag_document_relations => TagDocumentRelation.all
    }.to_json
  end

  get '/graphs/tags/tree/:id' do |tag_id|
    tag = Tag.find(tag_id)


    {
      :id => tag.id,
      :name => tag.name,
      :children => tag.documents.collect{|d|
        {:id => d.id, :name => d.name}
      }
    }.to_json

  end

  # end
```

Listing B.7: Rest Server / Search Module

Bibliography

[1] ESA, Statement of Work: Semantic Space Study, Reference = GSP-10/001-04-B-01-SEM, Issue 1, Revision 0, Date: 10-12-2009. 2

[2] Spring web services, 2005–2010. http://static.springsource.org/spring-ws/sites/1.5/. 5.3

[3] L Adamic and E Adar. How to search a social network. *Social Networks*, Jan 2005. 2.2.6

[4] Adobe Systems Incorporated. *Document management – Portable document format – Part 1: PDF 1.7*, 2008. 5.1.1

[5] Aduna B.V. User guide for sesame 2.3, chapter 7. sesame console, 2010. http://www.openrdf.org/doc/sesame2/users/ch07.html#section-console-repository-creation. A

[6] U Ai, L von Ahn, M Blum, and J Langford. Captcha: Using hard ai problems for security. *Proceedings of Eurocrypt*, Jan 2003. 5.3.4

[7] B Aleman-Meza, U Bojārs, H Boley, J Breslin, M Mochol, L Nixon, A Polleres, and A Zhdanova. Combining rdf vocabularies for expert finding. *The Semantic Web: Research and Applications*, pages 235–250, 2007. 5.2.4

[8] Kemafor Anyanwu, Angela Maduko, and Amit Sheth. Semrank: ranking complex relationship search results on the semantic web. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 117–127, New York, NY, USA, 2005. ACM. 5.2.4

[9] W Arms. Digital libraries. *books.google.com*, Jan 2001. 2.2.1

[10] Ken Arnold, James Gosling, and David Holmes. *Java(TM) Programming Language, The (4th Edition)*. Addison-Wesley Professional, 2005. 5.3

[11] G. Athanasopoulos, L. Candela, D. Castelli, P. Innocenti, Y. Ioannidis, A. Katifori, A. Nika, G. Vullo, and S. Ross. DELOS digital library reference model. Jan 2010. 2.2.5, 2.2.5, 3.8

[12] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. 2.2.6, 3.5.2, 4.1

[13] Keith Ballinger. *.NET Web Services: Architecture and Implementation with .NET*. Pearson Education, 2002. 5.3

[14] Krisztian Balog, Leif Azzopardi, and Maarten de Rijke. Formal models for expert finding in enterprise corpora. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 43–50, New York, NY, USA, 2006. ACM. 5.1.1

[15] R Baraniuk, V Cevher, M Duarte, and C Hegde. Model-based compressive sensing. *preprint*, Jan 2008. 7.2

[16] Philip A. Bernstein and Nathan Goodman. Multiversion concurrency control—theory and algorithms. *ACM Trans. Database Syst.*, 8(4):465–483, 1983. 5.2.1

[17] D. Bonino, F. Corno, L. Farinetti, and A. Bosca. Ontology driven semantic search. 2004. 5.2.4

[18] Dejan Bosanac. Apache XML-RPC adapter for spring, March 2006. http://www.oreillynet.com/onjava/blog/2006/03/apache_xmlrpc_adapter_for_spri.html. 5.3

[19] John G. Breslin, Stefan Decker, Andreas Harth, and Uldis Bojars. Sioc&#58; an approach to connect web&#45;based communities. *Int. J. Web Based Communities*, 2(2):133–142, 2006. 3.1.4

[20] D. Brickley and L. Miller. Foaf vocabulary specification 0.91. Jan 2007. 3.1.3

[21] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. *The Semantic Web — ISWC 2002*, pages 54–68, Jan 2002. 3.3.1, 5.3.2

[22] Mark H. Butler and James Rutherford. Distributed lucene: A distributed free text index for hadoop. May 2008. `http://www.hpl.hp.com/techreports/2008/HPL-2008-64.pdf`. 5.3.2

[23] D Byrne, C Murthy, S Shi, and C Shu. Lightweight directory access protocol (ldap) directory server cache mechanism and method. *US Patent 6,347,312*, Jan 2002. 5.2.4

[24] Hsinchun Chen. Semantic research for digital libraries. *D-Lib Magazine*, 5(10), October 1999. 2

[25] Le Chen, Lei Zhang, Feng Jing, Ke-Feng Deng, and Wei-Ying Ma. Ranking web objects from multiple communities. In *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 377–386, New York, NY, USA, 2006. ACM. 5.2.4

[26] S Cheung. Efficient video similarity measurement and search. *Citeseer*, Jan 2002. 7.2

[27] Hai Leong Chieu and Hwee Tou Ng. Named entity recognition: a maximum entropy approach using global information. In *Proceedings of the 19th international conference on Computational linguistics*, pages 1–7, Morristown, NJ, USA, 2002. Association for Computational Linguistics. 5.1.1

[28] J Cugini, S Laskowski, and M Sebrechts. Design of 3d visualization of search results: Evolution and evaluation. *Proceedings of SPIE, 2000*, Jan 2000. 7.2

[29] Steve Daly. Spring xmlrpcserviceexporter, February 2005. `http://blog.springsource.com/arjen/archives/2005/02/12/spring-xmlrpcserviceexporter/`. 5.3

[30] Fien De Meulder and Walter Daelemans. Memory-based named entity recognition using unannotated data. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003*, pages 208–211, Morristown, NJ, USA, 2003. Association for Computational Linguistics. 5.1.1

[31] Jeffrey Dean. Challenges in building large-scale information retrieval systems: invited talk. In Ricardo A. Baeza-Yates, Paolo

Boldi, Berthier A. Ribeiro-Neto, and Berkant Barla Cambazoglu, editors, *WSDM*, page 1. ACM, 2009. http://research.google.com/people/jeff/WSDM09-keynote.pdf. 1.1, 5.2.4

[32] Jeffrey Dean and Sanjay Ghemawat. Map reduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008. 3.7, 7.2

[33] H Dietze and M Schroeder. Goweb: a semantic search engine for the life science web. *BMC Bioinformatics*, 10 Suppl 10, 2009. 2

[34] Edsger W. Dijkstra. The structure of the "the"-multiprogramming system. In *SOSP '67: Proceedings of the first ACM symposium on Operating System Principles*, pages 10.1–10.6, New York, NY, USA, 1967. ACM. 5.2.1

[35] B Dom, I Eiron, A Cozzi, and Y Zhang. Graph-based ranking algorithms for e-mail expertise analysis. *Proceedings of the 8th ACM ...*, Jan 2003. 5.1.1, 7.2

[36] M Duarte, M Davenport, D Takhar, and J Laska. Single-pixel imaging via compressive sampling. *IEEE Signal Processing Magazine*, Jan 2008. 7.2

[37] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 613–622, New York, NY, USA, 2001. ACM. 5.2.4

[38] ECMA. *ECMA-334: C# Language Specification*. ECMA (European Association for Standardizing Information and Communication Systems), Geneva, Switzerland, fourth edition, June 2006. 5.3

[39] ECMA. *ECMA-376: Office Open XML File Formats*. ECMA (European Association for Standardizing Information and Communication Systems), Dec 2006. 5.1.1

[40] David Ellis and Ana Vasconcelos. Ranganathan and the net: using facet analysis to search and organise the world wide web. *Aslib proceedings*, 51(1), Jan 1999. 3.5.2

[41] Hui Fang and ChengXiang Zhai. Probabilistic models for expert finding. In *ECIR'07: Proceedings of the 29th European conference on IR research*, pages 418–430, Berlin, Heidelberg, 2007. Springer-Verlag. 5.1.1

[42] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, 2000. Chair-Taylor, Richard N. 3.1, 5.2.1

[43] Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang. Named entity recognition through classifier combination. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003*, pages 168–171, Morristown, NJ, USA, 2003. Association for Computational Linguistics. 5.1.1

[44] R Gaizauskas and Y Wilks. Information extraction: Beyond document retrieval. *Journal of documentation*, Jan 1998. 2.2.4

[45] Fausto Giunchiglia, Uladzimir Kharkevich, and Ilya Zaihrayeu. Concept search. In *ESWC 2009 Heraklion: Proceedings of the 6th European Semantic Web Conference on The Semantic Web*, pages 429–444, Berlin, Heidelberg, 2009. Springer-Verlag. 1.1

[46] Jim Gray. The transaction concept: virtues and limitations (invited paper). In *VLDB '1981: Proceedings of the seventh international conference on Very Large Data Bases*, pages 144–154. VLDB Endowment, 1981. 5.2.1

[47] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jaques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon. SOAP version 1.2 part 1: Messaging framework (second edition). 2007. `http://www.w3.org/TR/soap12-part1/`. 5.2.1

[48] James Hammerton. Named entity recognition with long short-term memory. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003*, pages 172–175, Morristown, NJ, USA, 2003. Association for Computational Linguistics. 5.1.1

[49] Jon Hartwick and Henri Barki. Explaining the role of user participation in information system use. *Manage. Sci.*, 40(4):440–465, 1994. 2.2.3

[50] Marti A. Hearst. Tilebars: visualization of term distribution information in full text information access. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 59–66, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co. 2.2.7

[51] Iris Hendrickx and Antal van den Bosch. Memory-based one-step named-entity recognition: effects of seed list features, classifier stacking, and unannotated data. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003*, pages 176–179, Morristown, NJ, USA, 2003. Association for Computational Linguistics. 5.1.1

[52] I. Herman, G. Melancon, and M.S. Marshall. Graph visualization and navigation in information visualization: A survey. *Visualization and Computer Graphics, IEEE Transactions on*, 6(1):24 –43, jan-mar 2000. 5.2.6

[53] P Heymann and H Garcia-Molina. Can tagging organize human knowledge? *ilpubs.stanford.edu*, Jan 2008. 5.2.5, 5.3.4

[54] M. Hildebrand, J. R. van Ossenbruggen, and L. Hardman. An Analysis Of Search-Based User Interaction On The Semantic Web. Technical Report INS-E0706, 2007. 3.5.2, 5.2.4, 5.2.6

[55] Christian H. Inzinger Schahram Dustdar Hong-Linh Truong, Johannes M. Schleicher. Esa itt number rfq 3-13016/09/nl/cbi semantic space study. Aug 2010. 1.4

[56] Andreas Hotho, Robert Jäschke, Christoph Schmitz, and Gerd Stumme. Information retrieval in folksonomies: Search and ranking. In *Proceedings of the 3rd European Semantic Web Conference*, volume 4011 of *LNCS*, pages 411–426, Budva, Montenegro, June 2006. Springer. 1.1

[57] ISO. *ISO/IEC 26300:2006: Open Document Format for Office Applications (OpenDocument) v1.0*. 5.1.1

[58] ISO. *ISO 15836:2009: The Dublin Core metadata element set*. International Organizations for Standardization, 2009. 3.1.2, 5.1.1

[59] Esther Kaufmann, Abraham Bernstein, and Renato Zumstein. Querix: A natural language interface to query ontologies based on clarification dialogs. In *In: 5th ISWC*, pages 980–981. Springer, 2006. 5.2.4

[60] Dan Klein, Joseph Smarr, Huy Nguyen, and Christopher D. Manning. Named entity recognition with character-level models. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003*,

pages 180–183, Morristown, NJ, USA, 2003. Association for Computational Linguistics. 5.1.1

[61] Jon Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM*, 46(5), Sep 1999. 3.6

[62] S Kruk, M Cygan, A Gzella, and T Woroniecki. Jeromedl: The social semantic digital library. *Semantic Digital Libraries*. 1.1, 4.1

[63] Sebastian Ryszard Kruk, Adam Gzella, Filip Czaja, WBultrowicz, and Ewelina Kruk. Multibeebrowse: accessible browsing on unstructured metadata. In *OTM'07: Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems*, pages 1063–1080, Berlin, Heidelberg, 2007. Springer-Verlag. 3.5.2, 4.1, 5.2.6, 7.2

[64] Sebastian Ryszard Kruk and Bill McDaniel. *Semantic Digital Libraries*. Springer Publishing Company, Incorporated, 2009. 2, 2.2.5, 3.2.1, 3.2.2, 3.5.3, 3.9.3

[65] Byron Y-L Kuo, Thomas Hentrich, Benjamin M . Good, and Mark D. Wilkinson. Tag clouds for summarizing web search results. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1203–1204, New York, NY, USA, 2007. ACM. 2.2.3, 3.5.2, 4.1

[66] Michal Laclavik, Martin Seleng, and Ladislav Hluchý. Towards large scale semantic annotation built on mapreduce architecture. In Marian Bubak, G. Dick van Albada, Jack Dongarra, and Peter M. A. Sloot, editors, *ICCS (3)*, volume 5103 of *Lecture Notes in Computer Science*, pages 331–338. Springer, 2008. 1.1, 3.7, 7.2

[67] Simon St. Laurent, Edd Dumbill, and Joe Johnston. *Programming Web Services with XML-RPC*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001. 5.2.1

[68] R Lee. Scalability report on triple store applications. *Massachusetts institute of technology*, Jan 2004. 3.3.1

[69] Michael Lesk. Understanding digital libraries. page 424, Jan 2005. 2

[70] Brian Loesgen, Andreas Eide, Mike Clark, Chris Miller, Matthew Reynolds, Robert Eisenberg, Bill Sempf, Srinivasa Sivakumar, Mike Batongbacal, Brandon Bohling, Russ Basiura, and Don Lee. *Professional Asp.Net Web Services*. Wrox Press Ltd., Birmingham, UK, UK, 2001. 5.3

[71] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008. 2.2.4

[72] E. Michael Maximilien. Web services on rails: Using ruby and rails for web services development and mashups. In *SCC '06: Proceedings of the IEEE International Conference on Services Computing*, page .39, Washington, DC, USA, 2006. IEEE Computer Society. 5.3

[73] Jeremy McAnally and Assaf Arkin. *Ruby in Practice*. Manning Publications Co., Greenwich, CT, USA, 2008. 5.3

[74] B McBride. Jena: Implementing the rdf model and syntax specification. *Citeseer*, Jan 2001. 3.3.1, 5.3.2

[75] C Middleton and R Baeza-Yates. A comparison of open source search engines. *Citeseer*, Jan 2007. 5.2.4

[76] Andrei Mikheev, Marc Moens, and Claire Grover. Named entity recognition without gazetteers. In *Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics*, pages 1–8, Morristown, NJ, USA, 1999. Association for Computational Linguistics. 5.1.1

[77] Zaiqing Nie, Yuanzhi Zhang, Ji-Rong Wen, and Wei-Ying Ma. Object-level ranking: bringing order to web objects. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 567–574, New York, NY, USA, 2005. ACM. 5.2.4

[78] L Page, S Brin, R Motwani, and T Winograd. The pagerank citation ranking: Bringing order to the web. *en.scientificcommons.org*, Jan 1998. 3.6

[79] R Ramachandran, S Movva, and S Graves. Ontology-based semantic search tool for atmospheric science. *... Processing Systems for ...*, Jan 2005. 5.2.4

[80] Leonard Richardson and Sam Ruby. *Restful web services*. O'Reilly, 2007. 5.3

[81] Cristiano Rocha, Daniel Schwabe, and Marcus Poggi Aragao. A hybrid approach for searching in the semantic web. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 374–383, New York, NY, USA, 2004. ACM. 7.2

[82] Tomas Salfischberger. Exporting spring beans with XML-RPC, March 2008. http://www.celerity.nl/blog/2008/03/exporting-springbeans-with-xml-rpc/. 5.3

[83] Bruce Schatz, William Mischo, Timothy Cole, Ann Bishop, Susan Harum, Eric Johnson, Laura Neumann, Hsinchun Chen, and Dorbin Ng. Federated search of scientific literature. *Computer*, 32:51–59, 1999. 2

[84] Michael F. Schwartz and David C. M. Wood. Discovering shared interests using graph analysis. *Commun. ACM*, 36(8):78–89, 1993. 5.1.1

[85] Marc M. Sebrechts, John V. Cugini, Sharon J. Laskowski, Joanna Vasilakis, and Michael S. Miller. Visualization of search results: a comparative evaluation of text, 2d, and 3d interfaces. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–10, New York, NY, USA, 1999. ACM. 5.2.6, 7.2

[86] Dongwook Shin, Hyuncheol Jang, and Honglan Jin. Bus: an effective indexing and retrieval scheme in structured documents. In *DL '98: Proceedings of the third ACM conference on Digital libraries*, pages 235–243, New York, NY, USA, 1998. ACM. 3.4.4

[87] Naveen Srinivasan, Massimo Paolucci, and Katia Sycara. An efficient algorithm for OWL-S based semantic search in UDDI. In *Semantic Web Services and Web Process Composition*, Lecture Notes in Computer Science, pages 96–110. 2005. 5.2.4

[88] T Staples, R Wayland, and S Payette. The fedora project. *D-Lib Magazine*, Jan 2003. 1.1, 3.2.2, 4.1

[89] M Szomszor, C Cattuto, H Alani, and KO'Hara. Folksonomies, the semantic web, and movie recommendation. Jan 2007. 2.2.3

[90] Valentin Tablan, Danica Damljanovic, and Kalina Bontcheva. A natural language query interface to structured information. In *ESWC'08: Proceedings of the 5th European semantic web conference on The semantic web*, pages 361–375, Berlin, Heidelberg, 2008. Springer-Verlag. 5.2.4

[91] T Tran, P Cimiano, S Rudolph, and R Studer. Ontology-based interpretation of keywords for semantic search. *The Semantic Web*, Jan 2008. 2.2.6, 5.2.5

[92] Thanh Tran, Philipp Cimiano, Sebastian Rudolph, and Rudi Studer. Ontology-based interpretation of keywords for semantic search. *The Semantic Web*, 4825:523–536, 2007. 5.2.4

[93] Martin Treiber, Hong-Linh Truong, and Schahram Dustdar. Soaf — design and implementation of a service-enriched social network. In *ICWE '9: Proceedings of the 9th International Conference on Web Engineering*, pages 379–393, Berlin, Heidelberg, 2009. Springer-Verlag. 2.2.3, 3.1.5

[94] M Twidale and D Nichols. Collaborative browsing and visualization of the search process. *Aslib proceedings*, Jan 1996. 5.2.6

[95] V Uren, P Cimiano, J Iria, and S Handschuh. Semantic annotation for knowledge management: Requirements and a survey of the state of the art. *Web Semantics: Science, Services and Agents on the World Wide Web*, Jan 2006. 3.2, 3.2.1, 3.2.2, 3.3

[96] Craig A. VanLengen and John D. Haney. Creating web services using asp.net. *J. Comput. Small Coll.*, 20(1):262–275, 2004. 5.3

[97] Viswa Viswanathan. Rapid web application development: A ruby on rails tutorial. *IEEE Software*, 25:98–106, 2008. 5.3

[98] L von Ahn. Games with a purpose. *Computer*, 39(6):92 – 94, Jun 2006. 3.2.1, 5.2.5, 5.3.4, 7.2

[99] T Vander Wal. Folksonomy. *Information Architecture Institute Members Mailing . . .*, Jan 2004. 2.2.3

[100] Craig Walls and Ryan Breidenbach. *Spring in action*. Manning Publications Co., Greenwich, CT, USA, 2007. 5.3

[101] S. Wasserman and K. Faust. *Social network analysis: Methods and applications*. Cambridge Univ Pr, 1994. 2.2.3

[102] Xiao-Yong Wei and Chong-Wah Ngo. Ontology-enriched semantic space for video search. In *MULTIMEDIA '07: Proceedings of the 15th international conference on Multimedia*, pages 981–990, New York, NY, USA, 2007. ACM. 7.2

[103] Felix Weigel. A survey of indexing techniques for semistructured documents. Jan 2002. 3.4, 3.4.1, 3.4.2, 3.4.4

[104] Wojciech Wiza, Krzysztof Walczak, and Wojciech Cellary. Periscope: a system for adaptive 3d visualization of search results. In *Web3D '04: Proceedings of the ninth international conference on 3D Web technology*, pages 29–40, New York, NY, USA, 2004. ACM. 7.2

[105] Dawit Yimam and Alfred Kobsa. Demoir: A hybrid architecture for expertise modeling and recommender systems. *Enabling Technologies, IEEE International Workshops on*, 0:67, 2000. 5.1.1

[106] J Yoon, V Raghavan, and V Chakilam. Bitcube: Clustering and statistical analysis for xml documents. *Journal of Intelligent Information Systems*, 2001. 3.4.4

[107] O. Zamir and O. Etzioni. Grouper: a dynamic clustering interface to Web search results. *Comput. Networks*, 31(11):1361–1374, 1999. 2.2.7

[108] Alireza Zarghami, Soude Fazeli, Nima Dokoohaki, and Mihhail Matskin. Social trust-aware recommendation system: A t-index approach. In *WI-IAT '09: Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, pages 85–90, Washington, DC, USA, 2009. IEEE Computer Society. 3.4.4

[109] GuoDong Zhou and Jian Su. Named entity recognition using an HMM-based chunk tagger. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 473–480, Morristown, NJ, USA, 2002. Association for Computational Linguistics. 5.1.1