h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Hochschule Darmstadt

- Fachbereich Informatik -

Terminology-Based Retrieval of Medical Publications

Abschlussarbeit zur Erlangung des akademischen Grades
Master of Science (M. Sc.)

vorgelegt von

Ulrich Beez

Referent(in)	:	*Prof. Dr. Bernhard Humm*
Korreferent(in)	:	*Prof. Dr. Bettina Harriehausen-Mühlbauer*

Ausgabedatum: *07.04.2015*
Abgabedatum: *05.10.2015*

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht. Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen. Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Heusenstamm, den 05.10.2015

Abstract

Die zunehmende Anzahl und Spezialisierung der medizinischen Veröffentlichungen zum Thema „schwarzer Hautkrebs“ stellt die behandelnden Mediziner vor ein große Herausforderung: Immer mehr Zeit muss investiert werden, um auf dem neuesten Stand der Forschung zu sein. Zeit, die Medizinern in den allermeisten Fällen nicht in diesem Maße zur Verfügung steht. Es besteht also die Gefahr, dass die für die Behandlung hilfreiche Information zwar existiert und abrufbar ist, jedoch nicht auf die Schnelle gefunden werden kann – und damit nicht dem Patienten zugute kommt.
Ziel der vorliegenden Masterarbeit ist es, Medizinern den Zugriff auf falladäquate und aktuelle Informationen zu erleichtern und so zu helfen, dass Patienten stets nach dem neuesten Kenntnisstand behandelt werden können.
Hierzu wird aus verschiedenen Fach-Ontologien zunächst ein Medizin-Vokabular bestehend aus ungefähr 46000 englischen Medizin-Fachbegriffen in sechs Kategorien zusammengestellt. Unter Nutzung dieses Medizin-Vokabulars werden aufgrund von Benutzereingaben automatisch passgenau Wörter für die Eingabefelder der Digitalen Gesundheitsakte angeboten. So wird das Ausfüllen der Digitalen Gesundheitsakte bei gleichzeitiger Normierung des genutzten Vokabulars erleichtert.
Im Nachgang ist es dann aufgrund des normierten Medizin-Vokabulars möglich, ohne weitere Benutzer-Interaktion nach für die Behandlung relevanter Medizinliteratur zu suchen.
Als Literaturquelle wird die „PubMed“ Datenbank der weltgrößten Medizinbibliothek - der U.S. National Library of Medicine – genutzt. Das Erzeugen der Abfragen an PubMed erfolgt über zwei Wege, zum einen über statische Regeln, zum anderen über ein maschinelles Lernen aufgrund von Benutzer-Feedback.
Basis für die Abbildung dieser Funktionalität ist die Auswahl der Ontologien und der Medizin-„Suchmaschinen“. Darauf aufbauend wird im Rahmen der Arbeit ein bestehendes System um zwei Module - Terminology Service und Literature Service - sowie passende Anzeige-Komponenten ergänzt. Bei der Planung und Programmierung dieser neuen Module wird Wert auf deren Erweiterbarkeit gelegt.

Abstract

The growing number and specialization of medical publications concerning malignant melanoma envisages physicians with severe problems. More and more time is asked to remain up-to-date, time they rarely dispose of. Though helpful information for the treatment exists and could be found the physician runs the risk to not find it in time to be of help to the patient.

This thesis aims at supporting physicians to be able to gain easy access to literature concerning the case in question and to relevant newest studies, to guarantee patients to be treated according to the latest standard of knowledge.

To ensure this, various special ontologies provide a vocabulary of approximately 46.000 medical technical terms in English divided up into six categories. By using this vocabulary the appropriate terms will automatically be offered for the input fields of the electronic health record (EHR). This facilitates filling in the EHR, and at the same time the used vocabulary is standardized.

A standardized terminology allows to search for relevant medical literature for the treatment without any further user interaction. As literature source, the "PubMed" database of the world's most complete medical library – The U.S. National Library of Medicine – will be employed. Two ways will gain answers from PubMed: Applying static rules or user feedback based machine learning.

The foundation for the implementation of the above-mentioned functionality is both the selection of ontologies and the search engines. Within the scope of the thesis an existing system is supplied with another two modules, the terminology service and the literature service, including fitting view components. On planning and programming these new modules great importance will be attached to their further extensibility.

Contents

List of Figures

Listings

List of Tables

Chapter 1

Introduction

This master thesis is written in context with the EU sponsored project "SemAntically integrating Genomics with Electronic health records for Cancer CARE - SAGE-CARE"
"The aim of the project is to bring together subject matter experts from the academic and non-academic sectors to create a holistic informatics platform for rapidly integrating genomic sequences, electronic health records (EHRs) and research repositories to enable personalised medicine strategies for malignant melanoma treatment."[1]

Personalized Medicine

According to K.K. Jain, "There is no officially recognized definition of personalized medicine", the word first appeared around the 1999 and terms relevant to the concept include Genomic medicine, Tailored therapy and lots more ([2] p.1). The U.S. Food and Drug Administration offers a description too: "The term 'personalized medicine' is often described as providing 'the right patient with the right drug at the right dose at the right time'" [3]. In an Executive Summary of the European Alliance for Personalised Medicine the term is explained as follows: "Personalised medicine looks at health information from single patients, using the new and detailed information now becoming available through advances in science and technology"[4]. As can be seen by those definitions, personalized medicine includes a variety of aspects. An additional description could be a more patient centered treatment using fitting techniques relying on an accurately collected and maintained EHRs, including the history backed up by appropriate technology. Concerning this thesis not all but two aspects are substantiated:

1. Maintaining individual EHRs using a fitting medical vocabulary.

2. Displaying additional literature during EHR review.

1.1 NSilico Lifescience Ltd.

NSilico Lifescience Ltd. is located in Cork, Ireland [5] and has 1 to 10 employees according to the social professional network Linked in [6]. The Company offers easy-to-use data management and analytics software for the life sciences and healthcare industries [7]. Due to NSilico's experience and expertise in areas of biology, computing, software-development and medicine the customers are provided with solutions which significantly increase the efficiency and accuracy of their work [7]. Customers of NSilico are e.g. clinics. Two products are currently offered:

1.2 NSilico Lifescience Cancer Care Products

Simplicity

Is promoted by NSilico as "The world's most easy-to-use tool for the management of raw-sequence data"[8]. The software offers to customers a responsive, user-friendly interface to order, display and organize gene sequencing analyses.

Simplicity MDT

Promoted as "NSilico's Intelligent Multi-disciplinary Team (MDT) Management software (Simplicity MDT™) is a customizable cloudbased platform that works on multiple devices (e.g. tablet, smart phone, laptop, desktop computer etc.)"[9]. The software offers capabilities to create electronic health records (EHRs), track their changes and annotate certain sections of the EHR for MDT meetings.

1.3 Structure of this Thesis

The Simplicity MDT software described in chapter 1, Introduction, lacks the features outlined in chapter 2, Problem Statement. While using the knowledge from chapter 3, Background, the solution design is described in chapter 4, Terminology Service and chapter 5, Literature Service which supports the implementation in chapter 6, Implementation. Then, in chapter 7 the evaluation is presented where all parts of the solution are compared to the requirements. In chapter 8, Related Work, further literature is reviewed. The last chapter, no. 9, comprises Conclusion and Future Work.

Chapter 2

Problem Statement

The problems to be solved are to implement a terminology that is fitting the medical problem domain and in addition display accurately EHR fitting and useful new medical literature to the physicians while using the software product. A solution needs to respect the project goals mentioned in chapter 1, Introduction, as well as the corresponding mission statements of the software described in subsection 1.1, NSilico Lifescience Ltd. More precisely the problem can be separated in functional and non functional requirements.

2.1 Functional Requirements

1. A for the medical context fitting terminology must be implemented and displayed conveniently to the physicians while entering forms of the EHR. At least two prototypes showing the behaviour in different contexts must be implemented to meet the requirement.

2. Useful and accurately fitting literature based on the EHR must be displayed to physicians.

3. All new interface components must be designed with usability in mind (they must not interfere with the overall user experience of the program).

2.2 Non Functional Requirements

1. The terminology must be offered instantly (below 1 s).

2. Collecting ten contextually relevant literatures and displaying them must not interfere with the normal EHR processing time during physician's review (below 5 min).

3. All new modules must be easy to maintain and extend.

Chapter 3

Background

3.1 Machine Learning

"Machine Learning is concerned with the study of building computer programs that automatically improve and/or adapt their performance through experience" ([10] p. V). A variety of tasks can be solved by machine learning, starting from handwritten digit recognition to stack value prediction. Machine learning is commonly distinguished into three fields ([11] pp. 650):

1. Supervised learning: By using example input data and corresponding output data, the mechanism learns to map the input to the output.
2. Unsupervised learning: Only input data is presented, the mechanism is left alone to find structure in the input data.
3. Reinforcement learning: The mechanism interacts with a dynamic environment to achieve a goal, caused by its interaction with the environment, its adapting, to accomplish the goal, given no feedback by a teacher.

The goal of machine learning in context with this thesis is the prediction of results, given an input set. The basis of the prediction is previously learned data, the mechanism is trained by input values and corresponding output values, leading to a supervised machine learning scenario. There exists several ways to implement supervised machine learning. Two of them are

Artificial Neural Networks

The underlying idea is inspired by biological nervous systems ([12] p. 175). Different structures of Artificial Neural Networks (ANN) are possible ([12] p. 189). The neuron is a shared concept, "[...] characterized by a set of connection strengths, a thresh- old, and an activation function" ([12] p. 177). If the input reaches a certain value, employing a threshold function, the neuron 'fires', producing an output which serves as input to other neurons or as output of the ANN. During learning the weights are adjusted.

Support Vector Machines

"The SVM approach computes the maximal margin hyperplane for a training set. In other words, the SVM optimal separating hyperplane is the one which separates the two classes by maximizing the distance to the closest point from either [...]" label [13]. During the learning phase the hyperplane is found. Depending on the distribution of the data to learn, a simple (e.g. linear style) hyperplane may be insufficient to separate the data. Kernel methods are used which "[...] implicitly map objects to high-dimensional feature spaces, and then directly specify the inner product there“ ([14] p. 570). A fitting solution depends on the base data. E.g. R.Berwick describes the usage of polar coordinates to find a hyperplane in circle style to separate the data [15].

However, the problem to be solved by supervised machine learning is a multi-label classification problem (MLC). The basic idea can be compared to identify the various fruits in a basket and not to be confused by recognizing one fruit only. According to Tsoumakas and Katakis MLC algorithms can be grouped in to two categories ([16] p. 12, pp. 667)

Problem Transformation

The problem is split into multiple single-label classification problems and by applying a wrapper mechanism, each of the single-label classifiers is taken into account, resulting in a multi-label classification.

Algorithm Adaptation

By adapting the learning algorithm to deal with multi-label classification directly, the classification problem is approached.

Label associations are learned directly by using the direct approach. The problem transformation approach needs to be designed to handle label associations ([16] p. 672, [17] p. 4, [18], [19]).

3.2 Medical Ontologies

The word 'ontology' carries distinct meanings in different communities and was created around the early 17th century ([20] p.425). In philosophy, going back to ancient Greece, it is used "[...] by the branch of philosophy which deals with the nature and structure of 'reality'" ([21] p.1), ([20] p.425). However, in computer science it has another, more formalized scope; there are several definitions ([22] p. 287), one of them explaining that an ontology
"[...]is a catalog of the types of things that are assumed to exist in a domain of interest D from the perspective of a person who uses a language L for the purpose of talking about D. The types in the ontology represent the predicates, word senses, or concept and relation types of the language L when used to discuss topics in the domain D"[23].
By comparing the two definitions: In IT an ontology is normally specific to a certain problem

domain while philosophy uses it for whole nature and reality ([22] p. 296).
While comparing an ontology to a dictionary, an ontology has in general more information features displayed by word relationships (synonym, broader, part of,...). In [24] the following "Semantic Stair" is presented, showing that the ontology provides the most semantic richness:

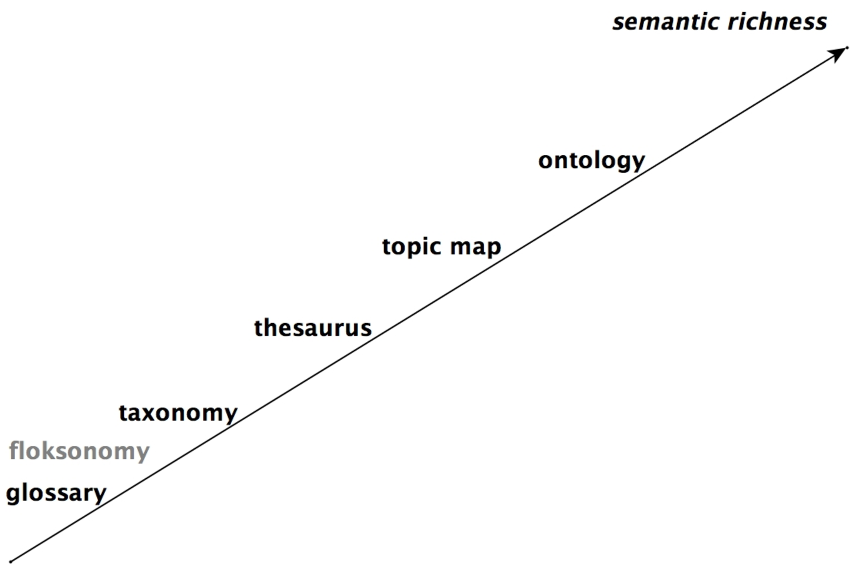


Figure 3.1: "Semantic Stair" (translation of A. Blumauer and T. Pellegrini)

However, "[...] in the life sciences, the term [...] ontology has also been more broadly defined as also comprising all forms of classified terminologies, including classifications and nomenclatures. Clinical ontologies provide not only a controlled vocabulary but also relationships among concepts allowing computer reasoning such that different parties, like physicians and insurers, can efficiently answer complex queries" ([25] p. 360). Hence for this thesis the word ontology is used to describe any kind of classified terminology in the specific medical domain.

3.3 Medical Literature Search Engines

These are used to search for medical literature, given an appropriate query. The corresponding query results depend on the data set the search engines have been provided with, and the way it has been indexed. Search engines use different literature bases. One prominent dataset used is the PubMed literature set. More details on the PubMed set can be found in appendix C.1, PubMed. The set of Medical literature search engines is collected using different sources: [26], [27]. To see the complete list of search engines investigated, please consult the appendix C, Medical Literature Search Engines

3.4 U.S. National Library of Medicine's PubMed API

Founded in 1836 and located in Bethesda, Maryland/USA, the National Library of Medicine (NLM) "[...] is a part of the National Institutes of Health, U.S. Department of Health and Human Services [...]" [28]. In 2008 NLM offered a collection of over 19 million medical literatures. According to NLM's web page its the world's largest medical library [28]. Associated to NLM is the National Center for Biotechnology Information (NCBI); here a variety of databases, resources, and APIs are offered [29]. One of the APIs provided is the Entrez Programming Utilities (E-utilities), offering access to 38 data bases, including PubMed [30].

3.4.1 PubMed API

A short description of PubMed can be found in appendix C.1, PubMed. Using a pipeline approach, it is possible to query PubMed by the previously mentioned E-utilities [31]. Querying PubMed with the E-utilities takes place in two phases and relies on HTTP requests. Each time the db parameter is set to 'PubMed'.

1. Relevant literature IDs are retrieved by handing over a search string to the 'esearch' endpoint and specifying the desired output format.

2. In using the literature IDs, the 'efetch' endpoint is accessed to retrieve literature information such as title, abstract, and author information, the desired output format can be specified.

3.4.2 API Usage

NLM offers a guide how to use the API at [32]. The following sentences are a short summary.

Disclaimer and Copyright

In case the PubMed API is used in a software, the users must be presented with NLM's Disclaimer and Copyright notice.

Availability

To not overload the API not more than 3 requests per second are recommended. In addition large jobs shall take place on weekends or between 9:00 PM and 5:00 AM Eastern time during weekdays. A failure to comply may result in an IP block. NLM encourages API users to register. After the registration the IP is unblocked [32].

Privacy

The communication to the API is not encrypted and the search terms are logged by the API. An additional API is offered to access the terms used for search which is used by researchers, e.g. Mosa and Illhoi do association mining of search tags in PubMed search sessions [33].

3.5 Apache Solr

Solr, a project of the Apache Software Foundation, is an open source enterprise full-text search server programmed in Java while partially using XML(Extensible Markup Language) ([34] p. 7). Solr offers also other features, a REST like service for CRUD (create, read, update, delete) operations on data and a web admin interface ([35], [34] p. 7). It is based on another Apache Project named Lucene, which "[...]provides Java-based indexing and search technology, as well as spell checking,[...] and advanced analysis/tokenization capabilities" [35].

Solr setup

The Solr distribution comes hand in hand with an embedded jetty (Webserver and Java servlet container) ([36] p. 5, [37]). Because jetty and Solr both rely on Java it's the only external dependency. However, Solr is not limited to use jetty, it is possible to deploy Solr e.g. on Apache Tomcat ([36] p. 10). Solr can be divided into two parts. Part one the Solr application a .war-file (web application archive [38]). Part two a folder storing the different configurations and the corresponding (Lucene) index files. The link between those two parts is the SOLR_HOME variable [39]. The variable can be set in the environment running Solr and needs to be changed if the setup differs from the default jetty installation.

Depending on the way Solr is incorporated into the application, Solr might require further configuration. Caused by the web's same-origin (security) policy where "[...] restrictions prevent a client-side Web application running from one origin from obtaining data retrieved from another origin" [40], a web browser may reject data from Solr. At least two solutions for the problem exist: Either enable CORS (Cross-Origin Resource Sharing) by modifying Solr's war file to include the required servlet filters or Solr needs to be queried differently e.g. with JSONP.

Configuring Solr

Solr offers functionality to run multiple cores in one instance. Each core could contain a different entity type ([34] p. 185). To put a Solr instance to work, at least the following configuration aspects should be considered for each core:

1. schema.xml
   Content of the file is the index structure definition ([36] p.28). Its content is roughly comparable to a datab base schema instead of rows, fields are declared. Here the unique key field (similar to rational databases primary key column), and for each field its name, data type and possible attributes (required, multi valued, indexed, ...) are configured ([34] p. 32). Solr allows dynamic fields with name

wildcards. Leading to an almost schema-less mode where fields can be added on demand ([34] p. 42). Dynamic fields chose their field type definition by matching the field name to the dynamic field name [41]. To interpret field content differently, a copy field can be defined ([34] p. 31). By defining a copy field, "[...]several distinct field types" can be applied "to a single piece of incoming information"[42]. Each field type behaviour during query and index time can be defined using analyzers containing tokenizers and filters [43]. Thus it is possible to define the following behaviour by changing the schema.xml only:

(a) 'Hello world' is to be stored e.g. unchanged in field 'original' and via copy to a field 'lower cased n-gram' which type is defined as follows:
(b) indicate with the analyzer of type index, that all of the following takes place on index time
(c) apply a tokenizer to split data on white space ('Hello world' becomes two entries 'Hello', 'world')
(d) additionally transform data to lowercase ('Hello' is changed to 'hello' while 'world' remains unchanged)
(e) and create n-grams ('hello' will be stored multiple times as 'h', 'he', 'hel', 'hell', 'hello' - the same applies for 'world')

The example results are 10 entries in field 'lower cased n-gram' because of the n-grams (5 entries from 'Hello' and 5 from 'world') each having the neighbour field 'original' containing 'Hello world'.
A detailed description of configuration parameters can be found in the Solr wiki: [44] as well as a list of analyzers, tokenizers and filters: [43].

2. solrconfig.xml
   This is the file where the solr core is configured. More precise, it contains the path to the (Lucene) index files, path to Solr addons, cache configurations, endpoint configuration for query (e.g. in which field to query by default) and insert/update and more [45]. The list of configuration options can be found at the Solr wiki: [45].

Solr security

By default the jetty servlet container running Solr listens to localhost only, hence the Solr web service is only locally reachable. Neither jetty nor Solr have any log-in mechanism which are enabled by default.
If Solr is accessible from the outside (e.g. jetty was reconfigured or Solr is running on another servlet container), any client connecting the server can access Solr's web service (and is able e.g. to flush the index).
Example URL to flush the Solr index of core1:

`http://localhost:8983/solr/core1/update?stream.body=<delete><query>*:*</query></delete>&commit=true` [46]
There are multiple ways to prevent Solr from unauthorized changes. Possible solutions to the problem are listed as follows:

1. Servlet filtering to black and white list Solr servlet endpoints
   By adding appropriate configured servlet filters to Solr's .war file, it is possible to limit access to Solr endpoints. E.g. all endpoints, but the one used to query Solr, can be configured to be accessible by certain IP addresses only. With the result that anyone can query Solr but only a specific IP addresse can change the index. A further extension to security on user level is possible by using a servlet filter allowing certain users or groups access to specific Solr endpoints.
2. Use a proxy to limit access
   Instead of exposing Solr itself to clients, a program acting as proxy between the client and Solr can be used to prevent unauthorized access. There are different proxies possible, two of them being:
   (a) Intelligent
   The proxy uses a client library to access Solr and offers its own (web) interface to clients. The proxy may contain logic to transform the query or reorder the result.
   (b) Forwarding only
   In general, the proxy type offers the same capabilities as described in black and whitelist solution using servlet filters. It basically forwards requests to certain Solr endpoints, while blocking others. However, the configuration takes place in the proxy. The actual potential of the solution is limited by the capabilities of the chosen web-server and it's re-write engine.

3.6 NSilico Lifescience Ltd. Simplicity Suite

The subsection describes the two previously mentioned products from NSilico and their context in more detail.

3.6.1 Simplicity

Currently ordering a gene sequencing analyses comes hand in hand with filling out a paper based order and sending it over to the laboratory. The laboratory then organizes the sample collection with the clinic. After collecting the sample, sequencing and analysis takes place. Those are extensive tasks, e.g., analysis alone covers matching certain gene configurations to known, relevant patterns. The whole process can be shortened by using Simplicity. It allows near instant sample collection ordering and sample re-usage for additional gene tests as well as managing raw-sequence data.

3.6.2 Simplicity MDT

At the moment, customers rely on paper based medical health records and physician's notes. Each record consists of different documents describing the health and treatments of an individual patient. During a MDT session, specialized and allied health professionals meet (surgeons, oncologists, dermatologists, nurses and other) to discuss treatment options of different patients [47]. To discuss patient health and possible treatments in MDT meetings using Simplicity MDT, detail patient data needs to be entered into the system.

Creating an EHR

Like its paper based relative the electronic version of the patient record consists of different areas. During EHR creation or change a user enters patient details into the application using distinct input fields. The user is presented with basic fields like gender and age to advanced fields like current medications to special conditional fields matching the current medical context, e.g. to detail a certain melanoma state or even a picture. Each input field uses its appropriate style: checkboxes to select a true/false value, dropdown lists to chose from different input options, long and short free text fields to insert words or file chosers to upload pictures. To have at least the same reliability as the paper version, all changes on an EHR are tracked automatically. Aside from different (conditional) input fields, the application cross checks some fields being related and provides a suggestion to the user. The result is a suggestion only and can be overwritten by the user.

One of the final steps during EHR creation or change is to schedule a discussion in a MDT meeting.

Usage of Simplicity MDT during the MDT meeting

During a MDT meeting all prior chosen EHRs are shown on a list view displaying a short summary per patient EHR. From the list view the whole EHR including all details, the change history and associated media (like pictures) can be viewed on by pushing a button. The view is the basis for MDT discussions and their results are entered in special input fields, annotating the EHR. Once all results are listed, the user can choose how to further treat the patient as suggested by the MDT, e.g., schedule the patient for another MDT meeting.

The basic technology stack (used database technology, programming languages for the core program as well as for the user front-end) is already decided. Only if it proves to be insufficient there is need to change it.

In more detail, the technology stack is based on Microsoft products, starting with a MS-SQL Server on database level to a core application written in C# interconnected with RAZOR MVC to provide the web front-end. The application is basically organized in a two layer architecture where the database is tightly coupled to view components enabling rapid changes on the data model with corresponding view changes. There are ongoing activities like, to further extend the application to incorporate more features and modifying the base application structure to a multi-layer architecture as well as changing the data presentation to a more generic model. All those activities are not in direct scope of the thesis. However, when features use parts of the application, the multi-layer idea and the generic data model are the choices to take. It will ease later migration. So, driven by multiple factors like the change of the application architecture, additional features, and maintenance, all new features need to be planned and implemented in a modularized way.

Chapter 4

Terminology Service

The terminology service is the key to for filling the terminology requirements described in chapter 2, Problem Statement. One major component the service uses is the term store described in subsection 4.2.5, Term Store. Combining the service and the term store, functionality for querying terms given a (partial) term and a context is provided. An example of the user interface is shown in figure 4.5, User Interface - Terminology Service - Semantic Autosuggestion. Prior to querying a vocabulary is established, different term categories are identified based on the EHR in section 4.2, Offline/ ETL Mode. Using those categories an ontology selection takes place in subsection 4.2.1, Ontology Selection, leading to a set of category fitting ontologies. Based on those ontologies the term 'store' is filled. The terminology service relies on accurate semantic mechanisms which are explained along the way (in subsections 4.2.2, Score-Based Ranking; 4.2.4, Duplicate Term Removal; 4.3.1, Semantic Result Optimization).

4.1 Overview

The diagram below shows the terminology service embedded in the Simplicity MDT application. In general, the service has two operation modes, shown as paths in the diagram and explained afterwards.

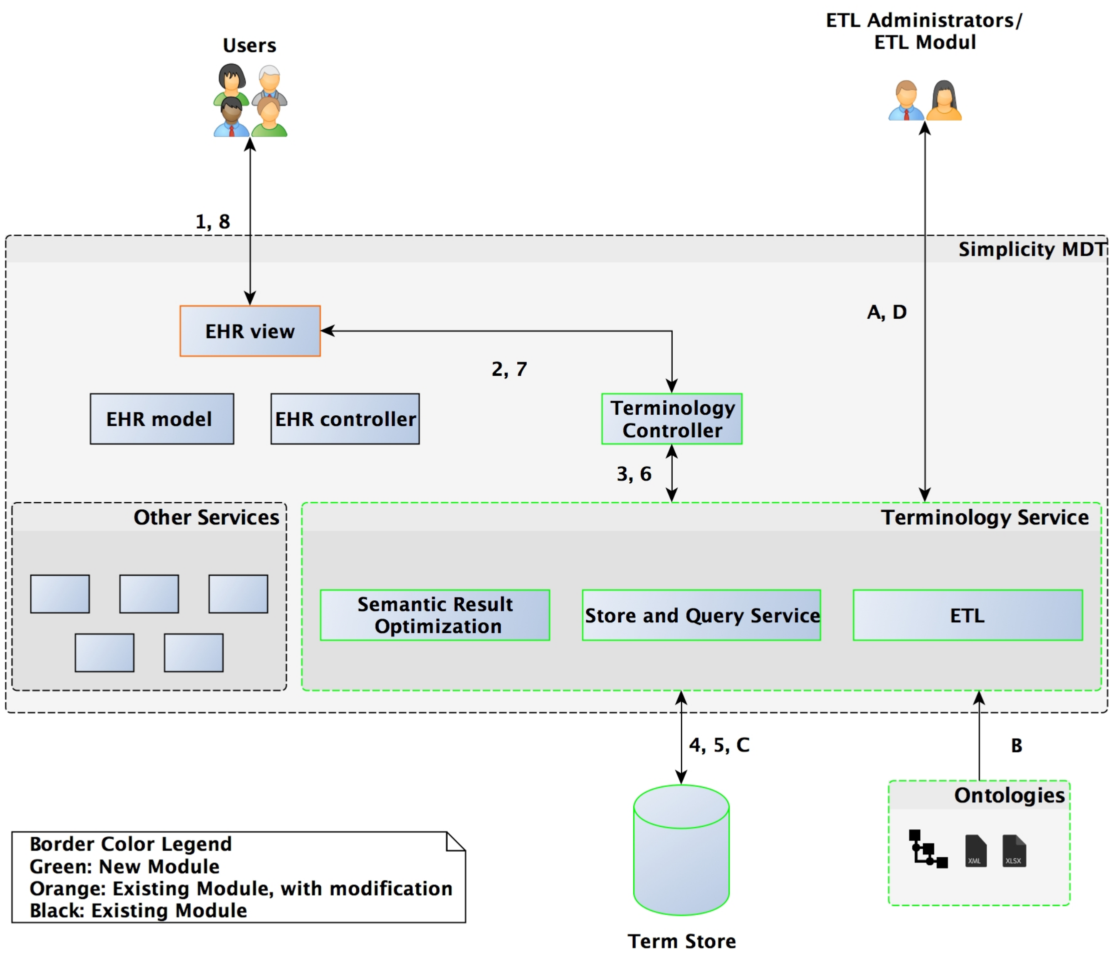


Figure 4.1: Overview Terminology Service

Online/query mode path with numeric labels (1-8):

1. A user starts to type, filling out an EHR form field. The following happens on each key press.
2. The EHR view routes the (partial) user input alongside the configured context and the amount of expected suggestions to the terminology controller.
3. Those informations are picked up by the terminology controller, its purpose being to map the request from the front-end format into an internal representation and to forward the request to the terminology service.
4. Using the user's input, the terminology service triggers a search in the term store.

5. As soon as the results are ready, the terms found are send to the terminology service.
6. After the duplicate removal and the semantic result optimization are completed, taking the provided context into account, the context fitting terms are send back to the terminology controller.
7. As the data arrives at the terminology controller, it is mapped to an appropriate view format and forwarded to the EHR view.
8. All context fitting terms, based on the partial user input, are presented to the user.

Offline/ ETL (Extract, Transform, Load) mode path with character labels (A-D):

(A) An ETL administrator or module triggers the terminology service for an ETL run.

(B) Terms from preconfigured ontologies are loaded.

(C) After duplicate removal and applying semantic filters the terms are sent alongside other information to the term store.

(D) To enable further processing of all terms send to the term store, they are then sent back to the invoker.

A detailed description of both operation modes can be found in the following sections 4.2, Offline/ ETL Mode and 4.3, Online/Query Mode.

4.2 Offline/ ETL Mode

The idea of a medical terminology service is connected to the EHR. An EHR contains different free text input fields, each describing a certain state of the patient's health. By studying the EHR, prominent fields are found:

1. medication used by the patient
2. treatments the patient received
3. diagnoses and findings by the physicians
4. patient's diseases
5. findings of the gene analysis
6. body parts where a melanoma occurs
7. other relevant health issues

Those different fields lead to six different categories of terms they can be filled with, being: Medication, Activity, Symptom, Disease, Gene and Anatomy.
As shown in the example field list, not all fields can be assigned to exactly one category, in addition, more categories could be added later. The field named "Other relevant health issues" can be filled with terms of any category. For each category there is need for a fitting ontology.

4.2.1 Ontology Selection

Eleven ontologies are preselected using different resources: [48] p 21; [49]; [27] pp. 9. The complete list of ontologies, each with a short description, can be found in appendix A, Medical Ontologies. The ontologies are available in various formats. However extractions of all ontologies but the Medical Subject Headings and the National Cancer Institute Thesaurus are offered at Ontobee [49] as '.Xlsx' files. Prior to the decision which ontologies are used, the ontologies are assigned to the categories. Than the selection criteria as well as the individual result for each ontology are described. In the following the ontologies are assigned to the categories.

1. Medication
   The Drug Ontology, National Drug File - Reference Terminology, Medical Subject Headings, National Cancer Institute Thesaurus

2. Activity
   Medical Subject Headings, National Cancer Institute Thesaurus

3. Symptom
   Symptom Ontology, Medical Subject Headings, National Cancer Institute Thesaurus

4. Disease
   Human Disease Ontology, Medical Subject Headings, National Cancer Institute Thesaurus

5. Gene
   Gene Ontology, Ontology of Genes and Genomes, Medical Subject Headings, National Cancer Institute Thesaurus

6. Anatomy
   Anatomical Entity Ontology, Foundational Model of Anatomy, Uber Anatomy Ontology, Medical Subject Headings, National Cancer Institute Thesaurus

Selection Criteria

Using four EHRs as test data, all fields are assigned to the previously mentioned categories. The following terms per category are used for checking.

1. Medication
   Warfarin, NOAC, Antiplatelet, Dacarbazine, Ipilimumab

2. Activity
   Malignant Melanoma removed, Melanoma Insitu removed, Lesion Excised, SNB, CXR, US, Bone scan, Serum LDH, Lymphadenectomy, CT, MRI, TETSCAN, Image Guided Biopsy, Ultrasound Study, Excision Biopsy, Re-excision Biopsy, Wide Excision for Margins, Incisional biopsy, Sentinel node biopsy, FNAC, Trucut Biopsy, Metastasectomy, Immunotherapy, Right Axillary Dissection, Pet CT Scan, CT Brain

3. Symptom
   Ulceration, Colitis

4. Disease
   Moderate Dementia, Severe Dementia, Primary Melanoma

5. Gene
   BRAF, V600E, EGFR

6. Anatomy
   Right Pinna, Head, Neck, Upper Limb, Lower Limb, Trunk, Right Arm, Left Leg, Thorax, Abdomen, Pelvis, Mid Back

Selection

The ontologies are consulted and checked for the previously identified vocabulary. The table showing the matches per term and ontology can be found in appendix A.1.

1. Medication - 5 terms
   National Drug File - Reference Terminology: 3 matches, The Drug Ontology: 4 matches, Medical Subject Headings: 5 matches, National Cancer Institute Thesaurus: 5 matches

2. Activity - 25 terms
   Medical Subject Headings: 11 matches, National Cancer Institute Thesaurus: 14 matches

3. Symptom - 2 terms
   Symptom Ontology: 1 match, Medical Subject Headings: 2 matches, National Cancer Institute Thesaurus: 2 matches

4. Disease - 3 terms
   Human Disease Ontology: 0 matches, Medical Subject Headings: 1 match, National Cancer Institute Thesaurus: 2 matches

5. Gene - 3 terms
   Gene Ontology: 1 match, Ontology of Genes and Genomes: 2 matches, Medical Subject Headings: 2 matches, National Cancer Institute Thesaurus: 3 matches

6. Anatomy - 11 terms
   Foundational Model of Anatomy: 1 match, Anatomical Entity Ontology: 2 matches, Uber Anatomy Ontology: 6 matches, Medical Subject Headings: 8 matches, National Cancer Institute Thesaurus: 9 matches

Observations during result collection:
There, term overlaps, e.g., 'Warfarin' is present in all ontologies of the 'Medication' category. Some ontologies outperform all other ontologies in one category for one term: e.g. in the 'Activity' category the term 'Incisional Biopsy' is only present in the National Cancer Institute Thesaurus. The ontologies fitting to all categories (Medical Subject Headings, National Cancer Institute Thesaurus) do not contain all expected sample terms.
It can be concluded that only by using all ontologies a comprehensive terminology can be established. Leading to the decision that all investigated ontologies are used to make up the term set. With respect to the literature service (described in the next chapter 5, Literature Service) and the decision to use PubMed as external literature store via its API (described in the subsection 5.2.7, Medical Literature Search Engine Selection) as well as the performance of MeSH in the above evaluation, MeSH is potentially a very good choice to include. However the decision to include all ontologies comes hand in hand with the need to rank ontologies of one category and handle duplicate terms.

4.2.2 Score-Based Ranking

Because the base ontologies may be exchanged or more ontologies added in the future as well as the fact that there are already multiple ontologies in one category, a ranking between the ontologies of one category is established. Basis for each score is the test data matching from the previous subsection 4.2.1, Ontology Selection. The ontology with the highest term coverage takes the sections best score. In case there are several ontologies with identical score the most comprehensive one wins. Score based ranking is used for multiple purposes:

1. In case one term occurs in the same category more than once (originating from different ontologies), the base score of the term in question is compared to decide which one is favoured.

2. As input value for the term base score, making it possible to by default boost all terms of one ontology over another in the same category.

4.2.3 Term Data Model

Every term with its ontology base boost (described in the previous subsection 4.2.2, Score-Based Ranking) is stored with the following additional information:

1. Its unique ID, coming from the ontology for further use in the application
2. The term category
3. The term source
4. If present all synonyms
5. If present all broader terms
6. If present a term definition

Leading to the following term class. With private attributes only. In addition to the terms 'unique source ID' another 'convenient ID' named 'term ID' is introduced. The basis data types used by the class are primitives (String, Double), enumerations or lists of one primitive type. Any method including the public get and set methods for each attribute are not displayed in the diagram.

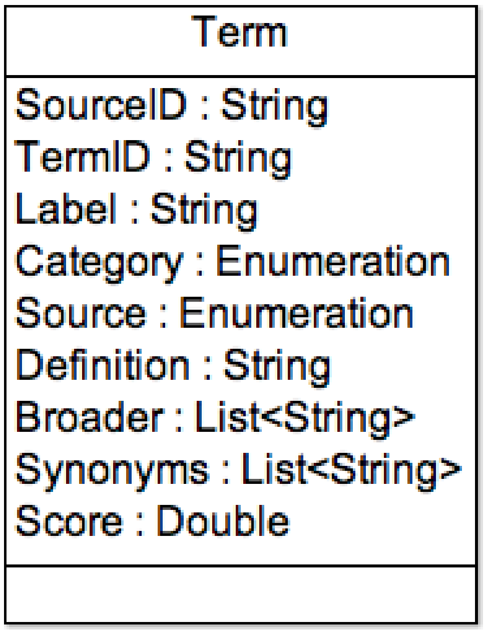


Figure 4.2: UML Class Diagram: Term

4.2.4 Duplicate Term Removal

Purpose of the duplicate term removal is to retrieve a set of term objects each having a unique ’Label’ attribute and taking the synonyms into account. The unique set is achieved by comparing a term to all other terms by two criteria:

Primary
: By Comparing the content of the ’Label’ attribute and asserting the synonyms.

Secondary
: By checking the term objects ontology score (described in 4.2.2, Score-Based Ranking) and its completeness in evaluating the presence of certain attributes. E.g. the presence of synonyms, broader term and the definition. All attributes of the ’Term’ class are described in subsection 4.2.3, Term Data Model.

As a match of the primary criteria occurs, the secondary criteria is taken into account.

4.2.5 Term Store

All terms acquired from the different ontologies are stored in a central term store enabled for very fast querying to respect the timing requirements. As there are a lot of terms, and the user will likely ask for terminology, providing an incomplete term more often than a complete term, the term stores query facility must be capable of handling partial queries fast. While performing a query, not only the term fields are inspected for a match but the synonyms of all terms as well. The results from the term store includes all term fields for later use. In an offline step, all terms of the previously identified ontologies are loaded with their base score.

4.2.6 ETL Process

The diagram shown below displays the ETL process. It consists of the three typical ETL steps extract, transform, and load, each marked using distinct boarder colors. In addition utility steps are displayed. All steps are described in detail following the diagram.

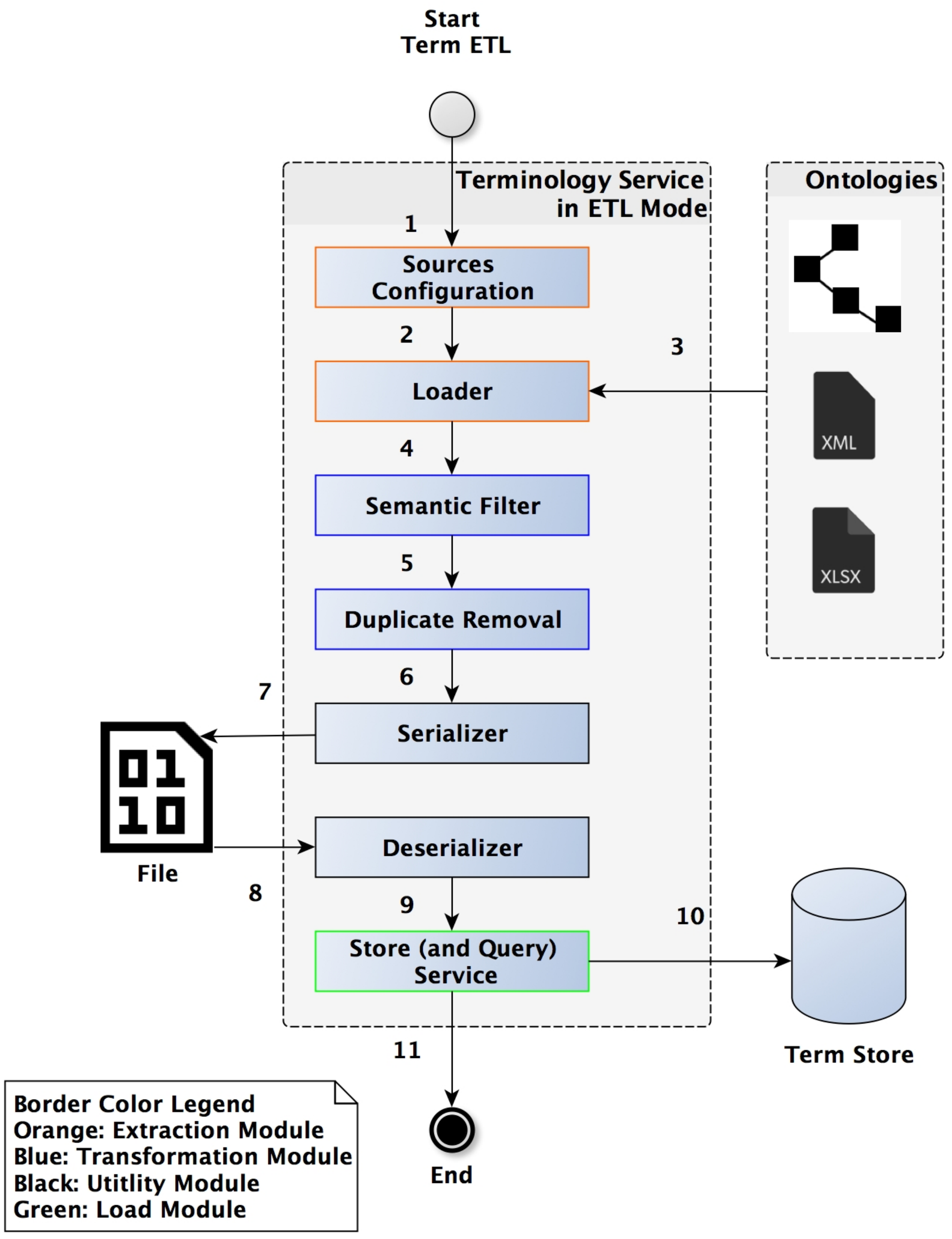


Figure 4.3: UML Activity Diagram: Term Store ETL

1. The terminology service ETL is triggered from the outside.

2. All source configurations residing in the configuration module are loaded with individual parameters for the appropriate loader per ontology. E.g.:
   For a MeSH subtree the MeSH loader is used given the Subtree entry point ID and an

ID exclude list as well as the target category. For the Xlsx formatted ontologies the Xlsx loader is used with filename, target category, ontology name and term exclude list.

3. Each source configuration is invoked, the excludes are applied and the mapping to the term class takes place, resulting in a list of terms.
4. All terms are send to the semantic filtering where a global exclude list is applied as well as term and base score manipulations take place. Examples: The term 'base score' can be manipulated using the term location depth in the ontology or the term frequency (term occurrence while mining text) or the term category. Some terms are "polluted" (have unwanted parts), those parts are removed.
5. Comparing all term labels, synonyms and taking the term completeness, ontology and category into account, duplicate terms are removed.
6. The set of duplicate free and semantic filtered terms are send to the serializer. Its task is to save them to a file.
7. Using a local file as target, the terms are saved in specific format.
8. Prior saved terms are loaded from the file.
9. The terms are handed over to the store (and query) service. Taking the base score into account, the storage of each individual term is triggered using the appropriate term store API.
10. All insert requests from the store (and query) service are consumed by the term store.
11. To signal the process end and allowing further term processing, the list of all loaded and stored terms is returned to the invoker.

With growing numbers of ontologies (and terms) the ETL processing time increases, making a full ETL unfavorable during application start. In the diagram shown above, a file is displayed (step 8, 9). The file decouples the slower extract and transform part from the faster loading part.

4.3 Online/Query Mode

Taking the six categories from the EHR identified in the beginning of section 4.2, Offline/ETL Mode, as basis, the context of an EHR field can be defined. The term suggestion process in the Online/Query Mode is shown in figure 4.4 and afterwards described in detail.

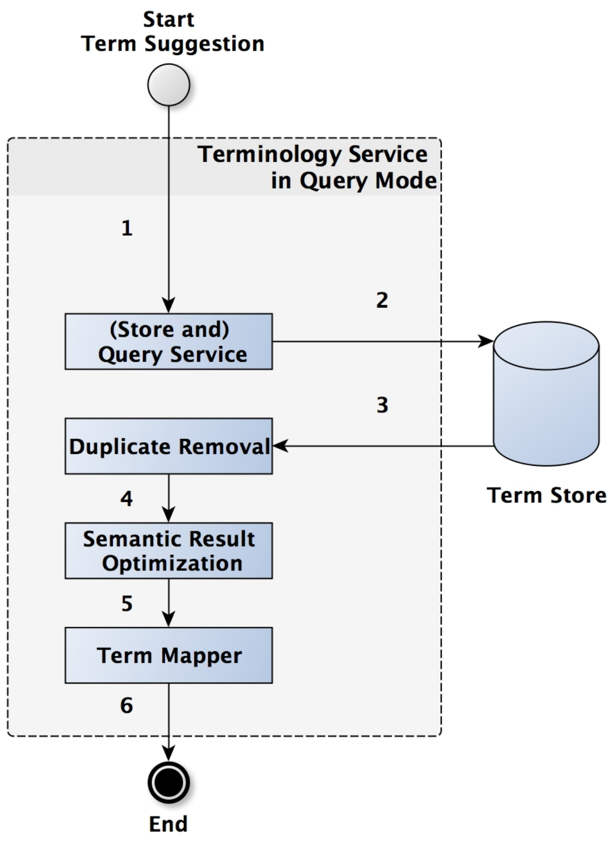


Figure 4.4: UML Activity Diagram: Querying the Terminology Service

1. The terminology service is externally invoked, asking for an amount of terms given a term (part) and its context.
2. Using the (store and) query service the request is forwarded to the term store.
3. After the term store has performed the lookup using partial matching on the term field as well as the synonym field, returning all fitting terms with their corresponding synonyms, broader terms, descriptions, category and ID. The results are handed over to the duplicate removal.
4. As soon as the results are cleared from any duplicates, using the default base score taking each category and term source into account, the matching terms are given to the semantic result optimization.

5. While using the original requests context and the found terms, the semantic result optimization takes place. Its final step is to re-arrange the search result.
6. Prior to sending all context fitting terms back to the invoker, each term representation in term store format is mapped to the output format.

4.3.1 Semantic Result Optimization

In general per term a multidimensional optimization problem needs to be solved. Dimension can be context specific to a query and static to a certain term. Some dimensions are listed below.

Context Specific

1. place in the term where the first match occurs
2. match count
3. category match and number of free slots in the result set for a certain category
4. user's preference

Term Static

1. term length
2. category default boost

Depending on the expected result the value of each dimension can be weighted. As final step the results are ordered accordingly.
In the 'terminology service' adoptions of the above mentioned dimensions are used. On service invocation, the maximum count of terms as well as a weight for each category (context) alongside the term (part) is provided.
As soon as the result set from the term store arrives each term's score is modified based on where the match occurs and wether or not the found term consists of multiple words. Terms consisting of one word, starting with the search term receive the highest boost.
All categories are sorted in descending order by their given weight and per category the term count is calculated. Prior to returning the terms, one of the following filters is applied:

Category Diversity

Starting with the highest weight alternating between all categories one term per category is added to the result list. If a category's weight does not allow for more terms to be added, the category is skipped.

Example: Three terms are requested for category medication with weight 2 and disease with weight 1, the result is the following list:

1st match from medication category

1st match from disease category

2nd match from medication category

Category Clusters

Instead of alternating the categories after each term (as category diversity does), the terms belonging to one category are outputted together.

Example (identical configuration as the category diversity example):

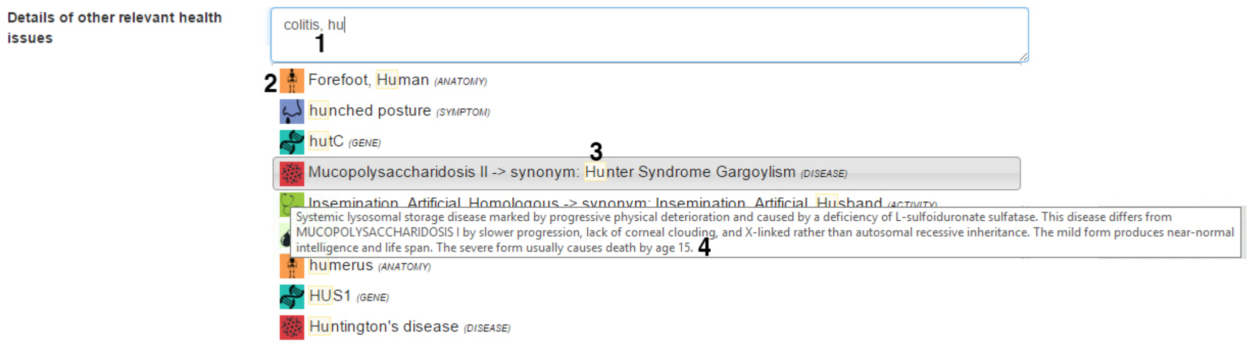
1st match from medication category

2nd match from medication category

1st match from disease category

4.3.2 User Interface

The picture below shows the terminology service's user interface part during EHR form filling. Four special features of the view component are tagged in the picture and explained afterwards.


Details of other relevant health issues

colitis, hu

1

2 Forefoot, Human (ANATOMY)

hunched posture (SYMPTOM)

hutC (GENE)

3

Mucopolysaccharidosis II -> synonym: Hunter Syndrome Gargoylism (DISEASE)

Insemination, Artificial, Homologous -> synonym: Insemination, Artificial, Husband (ACTIVITY)

Systemic lysosomal storage disease marked by progressive physical deterioration and caused by a deficiency of L-sulfoiduronate sulfatase. This disease differs from MUCOPOLYSACCHARIDOSIS I by slower progression, lack of corneal clouding, and X-linked rather than autosomal recessive inheritance. The mild form produces near-normal intelligence and life span. The severe form usually causes death by age 15. 4

humerus (ANATOMY)

HUS1 (GENE)

Huntington's disease (DISEASE)

Figure 4.5: User Interface - Terminology Service - Semantic Autosuggestion

1. Multi Word Problem
   As mentioned at the very beginning of chapter 4, Terminology Service, an EHR has fields where multiple terms can be entered. Using a term delimiter, to separate newly entered terms from old ones, the view component shows terminology for the last user input only.

2. Category Icons
   To increase the user experience an icon is displayed per term, indicating its category.

3. Highlighting the User Input
   The term (part) the user enters is highlighted in the suggested terms and synonyms.

4. Description
   Per term its description is displayed in a tooltip.

Chapter 5

Literature Service

The literature service is the answer to the literature problem stated in chapter 2, Problem Statement. It queries a literature store to find EHR fitting and helpful literature which is displayed to the user. Before a query to the literature store can be submitted, the query needs to be carefully worded. To create an appropriate EHR fitting query, the semantic EHR field mapping takes place as described in subsection 5.2.1, Semantic EHR Field Mapping. After mapping, all fields are handed to the Semantic Query Term Identification described in subsection 5.2.2, which identifies the search terms to enable query creation explained in subsection 5.2.5, Query Creation. As literature store an external source is used, the evaluation and decision process which store to used is done in subsection 5.2.7, Medical Literature Search Engine Selection. A screenshot of the interface presented to the user is shown in subsection 5.2.8, Displaying Literature.

5.1 Overview

The diagram below shows the literature service embedded in the Simplicity MDT application. To accomplish its task the service uses on major external dependency, the literature store. Its main purpose, displaying relevant literature, is shown as path with numeric labels (1-8). The path with character labels (A-D) shows how user feedback for certain literature and EHR is collected. All user feedback is saved in a feedback store. Following the diagram both paths are explained in more detail.

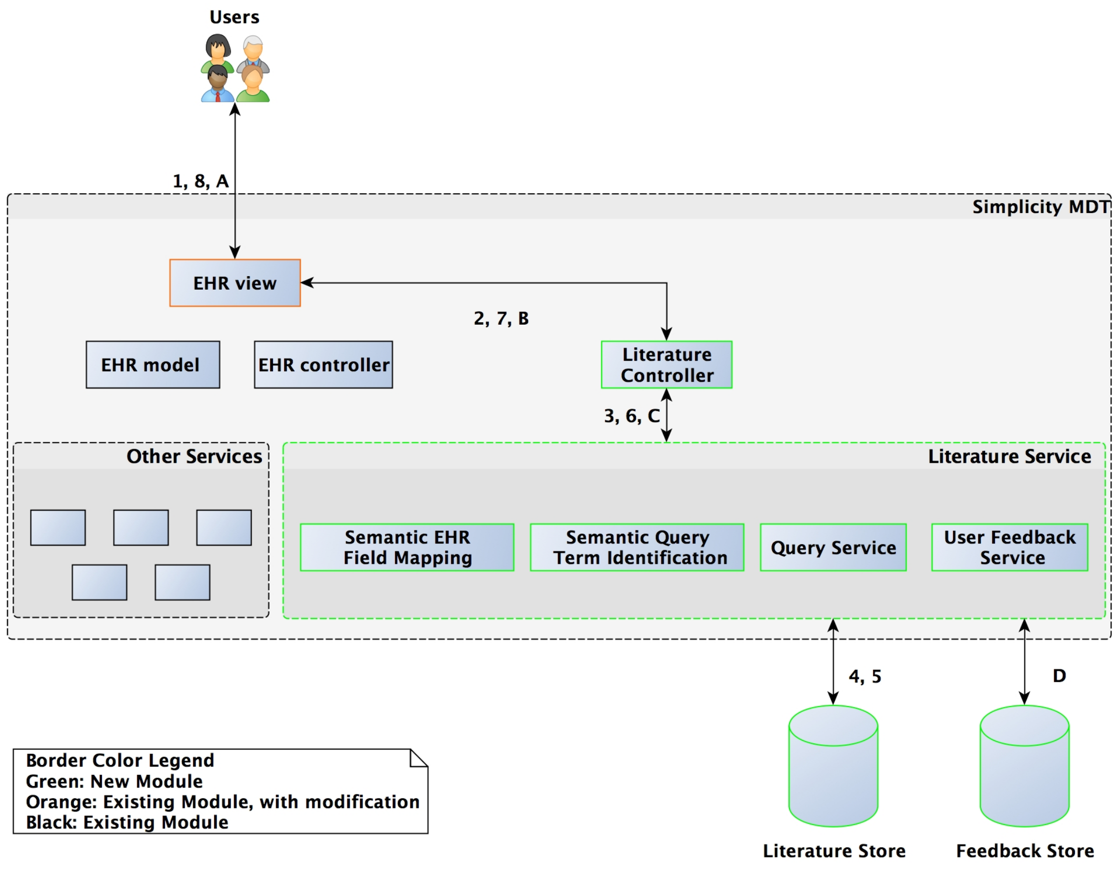


Figure 5.1: Overview Literature Service

Literature search path with numeric labels (1-8):

1. A user opens an EHR for review, as the front-end loads, literature is requested in the background, asynchronously using the EHR ID.
2. The EHR view forwards the request with the EHR ID to the literature controller.
3. Using the EHR ID, a lookup for the EHR takes place. The EHR is handed over to the literature service.
4. After semantic EHR field mapping, semantic query term identification and query creation, the literature service queries the literature store.

5. As the query reaches the literature store, medical literature is searched using the given query. Right after the results appear, they are sent back to the literature service.
6. The found literature is received by the literature service. All literature is mapped to an internal representation and send to the literature controller.
7. Converting the result to a view format on the way, the literature controller sends the literature to the EHR view.
8. A part of the EHR view is updated, taking the literature as input. EHR fitting literature is displayed to the user.

User Feedback path with character labels (A-D):

(A) While reviewing the literature for an EHR a user decides to up or down vote a certain literature. The voting feature is activated by the user and the view is updated directly.

(B) The user input is processed by the EHR view collecting the EHR ID and the literature ID. Those information is forwarded to the literature controller.

(C) After the EHR lookup using the provided EHR ID takes place, the tuple of EHR and literature ID is send to the literature service.

(D) All EHR data after the semantic EHR field mapping and literature search term extraction are send to the feedback store.

5.1.1 Literature Class

Output of the literature service is a list of Literature class instances. Throughout the literature service the class is used. Below, the class is described in more detail.
All attributes have private visibility and are excessed via public get and set methods. No methods are shown in the diagram. A short description of each attribute can be found following the diagram.

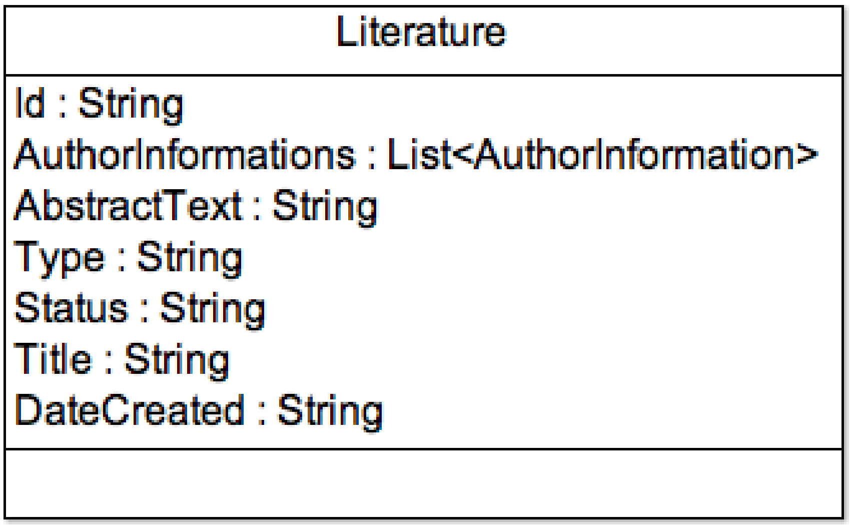


Figure 5.2: UML Class Diagram: Literature

The 'Id' attribute holds the literature ID used by the literature store. 'AuthorInformations' is a list of AuthorInformation being a complex type that contains the name and affiliation of an author. 'AbstractText' is of type string and contains the literature's abstract. 'Type' indicates of which type the literature is (e.g. article). The field 'Status' holds the status of the publication (e.g. preprint). 'Title' contains the title of the literature as string type. 'DateCreated' holds the creation date, it is of type string because of different date formats.

5.2 Literature Search

The UML activity diagram below shows the literature service during a literature search call. In general the services input data is an EHR and the output is a set of useful and fitting Literature classes in context of the EHR. Following the diagram each box, representing a component, is described.

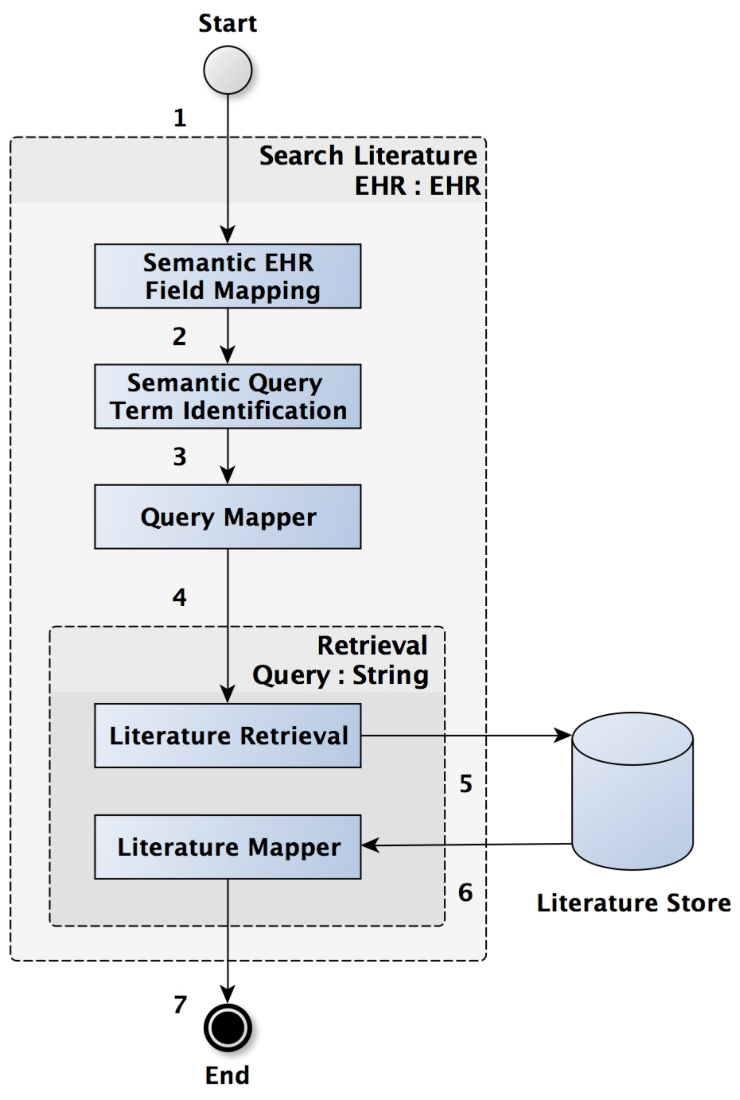


Figure 5.3: UML Activity Diagram: Literature Service - Search

1. On invocation a whole EHR is provided to the literature service. The internal processing begins with the semantic EHR field mapping, its task is to map the EHR to a service internal representation for further processing, taking care of multivalued fields and converting datatypes as needed.

2. After the mapping has taken place, the mapped EHR is send to the semantic query term identification component. It analyses the internal representation and prepares the

data for the query mapper. A lot of decisions take place, e.g., which fields are important and which are not. The output is a list of query terms.

3. The query mapper's task is to generate an appropriate query which can be processed by the literature store.
4. As soon as the query arrives at the literature retrieval component it is forwarded to the literature store using its API.
5. The query is send to the literature store. It is picked up by the store and processed.
6. When the results appear they are sent back to the literature mapper, which transforms the query results to the literature service's output format.
7. All found literature is provided to the invoker

5.2.1 Semantic EHR Field Mapping

The general idea of the Semantic EHR Field Mapping is related to the concept of loose coupling and can be summarized in four statements:

1. It provides an abstract view on the EHR by grouping fields containing data of the same category. e.g All medication fields are grouped to the medication category.
2. The datatype of individual field types are standardized. E.g. Two medication fields, one of type free text, the other a checkbox, are transformed into strings.
3. As the EHR evolves, both the field amount and individual field types change. E.g. A new treatment type is discovered, resulting in a new field and datatype in the EHR. By adding the field to the treatment cluster, the new field can be introduced to the literature service.
4. The EHR is changed more often than the literature service. E.g., the complete EHR representation is changed to a more general model. From the literature service perspective, the Semantic EHR Field Mapping is the only component in need to be altered.

Per EHR field an instance of the following SearchField class is created. The component's output consists of a list of SearchField classes.

SearchField Class

All attributes of the SearchField class are private. Public visible methods for getting and setting an attribute are not shown in the diagram. The attribute named FieldName with type enumeration is the category of the 'SearchField', e.g., 'Medication'. The Type attribute is an enumeration to indicate the actual type of the Value attribute. E.g., for 'Medication' the Type attribute is set to ’string’ and the corresponding value type is a string filled with the actual medication ’Warfarin’.

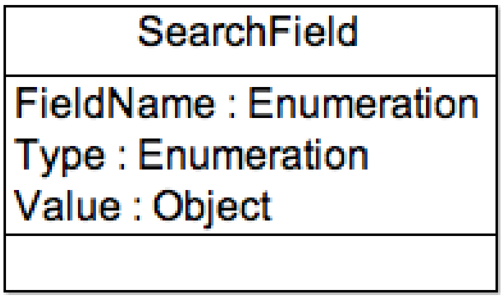


Figure 5.4: UML Class Diagram: SearchField

5.2.2 Semantic Query Term Identification

For the literature search some EHR fields are more important than others. E.g. searching for literature using the patient’s name will unlikely result in fitting and useful literature in contrast to searching for the patient’s melanoma stage.
The combination of the EHR fields and their interconnections influence the search result. Using all EHR fields and connect them using the logic "and" operator describes the EHR best, but is too specific. In the worst case no search results will be found. Employing the "or" operator the result is too general.
By enriching specific search fields with additional information or replacing a whole term, the probability of helpful search result increases. Examples are: Instead of searching for a medication, the medication concatenated with "side effects" is searched. Broad terms are replaced with narrow terms, "immunotherapy" is replaced with "Ipilimumab". All above-mentioned techniques can be applied using a rule or machine learning approach or a combination of both. The output is a list of search terms fit for further processing by the query creation module.

5.2.3 Rules

Using EHRs and corresponding expected literature search results, found by an expert, hard coded rules are created. By the static nature of those base data, the rule’s performance is acceptable for known EHRs but for unknown EHRs their performance will most likely be poor. A possible solution is the creation of more rules. However, the rule creation is a task not fit for end users. To compensate the behavior a more dynamic approach, facilitating machine learning is applied.

5.2.4 Machine Learning

The idea of machine learning in the problem domain implies:

1. To replace the previously mentioned rules by a long shot.
2. Use the output of the semantic field mapper for known EHRs and the corresponding literature search terms as input for machine learning, based on user feedback.

By training the mechanism using output of the semantic field mapper and providing corresponding literature search terms, it can be expected that at some point the predictions offered by machine learning will be sufficient.
To enable machine learning there is need for search terms per literature. Using the vocabulary created in chapter 4, Terminology Service, each literature is verified for terms. The combination of literature search terms (as expected output) and output of the semantic field mapper (as input) facilitate machine learning.

5.2.5 Query Creation

All output of the Semantic Query Term Identification (described in subsection 5.2.2) is combined into one query. The query syntax complies to the expected format of the literature store. Instead of implementing an own literature store an appropriate search engine in the internet is used. Prior to connecting the search engine, there is need to find one.

5.2.6 Medical Literature Search Engine Comparison

A study of 17 different medical search engines takes place. Each search engine employs a specific query language. Some differences are:

1. Whitespace or comma interpretation using logic operators, e.g. a query with two search terms (male dementia) is interpreted with “and” resulting in a search for male and dementia.
2. How multiword search terms are entered, e.g. to search for a multiword term (invasive malignant melanoma) it needs to be surrounded by brackets or quotation marks.
3. Special features of the search engine like ontology lookup and enrichment using more specific terms. E.g. instead of searching for the term melanoma the search engine queries for melanoma or (Hutchinson’s Melanotic Freckle) or (Melanoma, Amelanotic) or (Melanoma, Experimental).

Those factors are considered while creating the 7 queries used for comparison. All 7 queries are listed in appendix B, Medical Literature Queries. However, while performing tests, no demo-data is available. Hence, the test queries are formed by using the EHR form field description and inserting additional domain knowledge from an internet source [50]. The data of all search engines is captured in two periods:

1. April 23rd till April 29th, 2015
   PubMed, Textpresso, HubMed, Medlineplus, Quertle, ScienceDirect, Medscape, BibliMed, ASTIS, DOAJ, Google Scholar, HighWire, GoPubMed,POLS

2. May 4th 2015
   LIVIVO, BioMed Central, Pubget

A table showing additional search engine criteria (base data, usage license and API) can be found in C, Medical Literature Search Engines.

Search Engine	**Query 1**	**Query 2**	**Query 3**	**Query 4**	**Query 5**	**Query 6**	**Query 7**
PubMed	44	1	0	0	7112732	7112732	7135097
Textpresso	2908	0	0	0	error	error	error
HubMed	44	1	0	0	7112732	7112732	7135097
Medlineplus	0	4	0	0	78	3673	5093
Quertle	156	4905	0	0	0	0	0
ScienceDirect	1229	4	0	0	113682	2111864	2250319
Medscape	0	64	0	7	2	2	2
BibliMed	44	0	0	0	7112732	7112732	7135097
ASTIS	0	0	0	0	error	error	error
DOAJ	42772	55348	1377485	1340502	1376652	1376651	1378377
Google Scholar	8390	51000	0	0	288*	288*	288*
HighWire	1178	3431	7*	6*	1*	6*	6*
GoPubMed	44	1	0	0	100000	100000	100000
POLS	1	˜110	0	0	0	0	0
LIVIVO	54	47	0	0	7071996	7071996	6768505
BioMed Central	0	435	0	0	25	77544	82944
Pubget	0	35	0	0	7316437	1543688	1595157

Table 5.1: Literature Search Engine Results

Comment on table data

error	During query the web browser returns an error message (like HTTP error 500). No results are shown.
*	The query is cut by the search engine.

˜ No result count is shown by the search engine. The amount is calculated by multiplying the result pages with the number of results shown on the first page.

5.2.7 Medical Literature Search Engine Selection

The relevance of the search results is the most important metric while comparing different search engines. By the problem domain's nature the search result relevance to an EHR can be assessed by domain experts only, e.g. physicians. Search engine comparison by result quantity as shown in table 5.1, Literature Search Engine Results is no quality measurement. However, there are additional but weaker indicators about a search engine's usability:

1. Five of the seventeen search engines (PubMed, HubMed, BibliMed, LIVIVO, Pubget) return almost the same amount of results for one query (Query 5), because they rely on NLM's literature set.
2. Because the base data for the search originates from the EHR, it can be assumed that if a search engine is chosen which is already using a large amount of the terms offered by the terminology service for indexing, the search result quality will be better than using a search engine which is using another base terminology.
3. Caused by the need to integrate the search engine into the application an API can be useful.

Those weaker indicators lead to the decision of using NLM's PubMed, because

1. It is the most direct interface to NLM's literature set.
2. Aside from the search engine, NLM offers the MeSH vocabulary (more details on MeSH can be found in appendix A.11, Medical Subject Headings).
3. NLM offers an API to the search engine (see subsection 3.4.1, PubMed API).

5.2.8 Displaying Literature

Basis for displaying literature is the literature class. Figure 5.5 shows one element of the literature user interface. The complete interface has many of those elements, one per literature object. Not all but important information of each literature is displayed by default. Complete author details are shown after a user's interaction only (mark 1). Most important fields consists of literature title and abstract (mark 2 and 3). Using the literature's ID, a link to the literature's homepage at NLM is shown to the user (mark 4).
For each found literature in addition to the literature class attributes an up and down voting

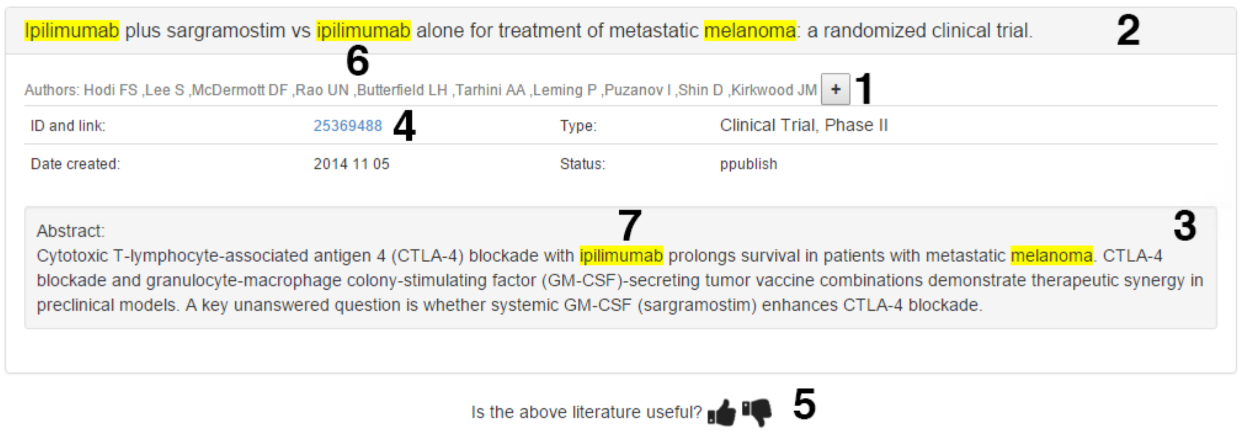
button is displayed (mark 5). By voting a certain literature up or down, training data for the machine learning component is created.

Asynchronous Literature Loading

To ensure the best user experience and meet the maximum time requirement a literature search is triggered asynchronously as soon as the user opens the EHR for review. While the back-end handles the actual search, the user inspects the EHR data. As soon as the literature search finishes, part of the EHR is updated, and the context fitting literature is displayed.

Literature Search Term Highlighting

By highlighting the search terms used for literature acquisition, a user can inspect the literature easily. All search term occurrences in each literature's title and abstract are highlighted (mark 6, 7).



Ipilimumab plus sargramostim vs ipilimumab alone for treatment of metastatic melanoma: a randomized clinical trial.

Authors: Hodi FS ,Lee S ,McDermott DF ,Rao UN ,Butterfield LH ,Tarhini AA ,Leming P ,Puzanov I ,Shin D ,Kirkwood JM

ID and link:	25369488	Type:	Clinical Trial, Phase II
Date created:	2014 11 05	Status:	ppublish

Abstract:
Cytotoxic T-lymphocyte-associated antigen 4 (CTLA-4) blockade with ipilimumab prolongs survival in patients with metastatic melanoma. CTLA-4 blockade and granulocyte-macrophage colony-stimulating factor (GM-CSF)-secreting tumor vaccine combinations demonstrate therapeutic synergy in preclinical models. A key unanswered question is whether systemic GM-CSF (sargramostim) enhances CTLA-4 blockade.

Is the above literature useful?

Figure 5.5: User Interface - Literature Service

5.3 User feedback

The user feedback is relevant to the machine learning approach mentioned in subsection 5.2.4, Machine Learning. Using the user feedback, training data for machine learning is created. The UML activity diagram in figure 5.6 shows the literature service during a user feedback call. Each step is explained after the diagram.

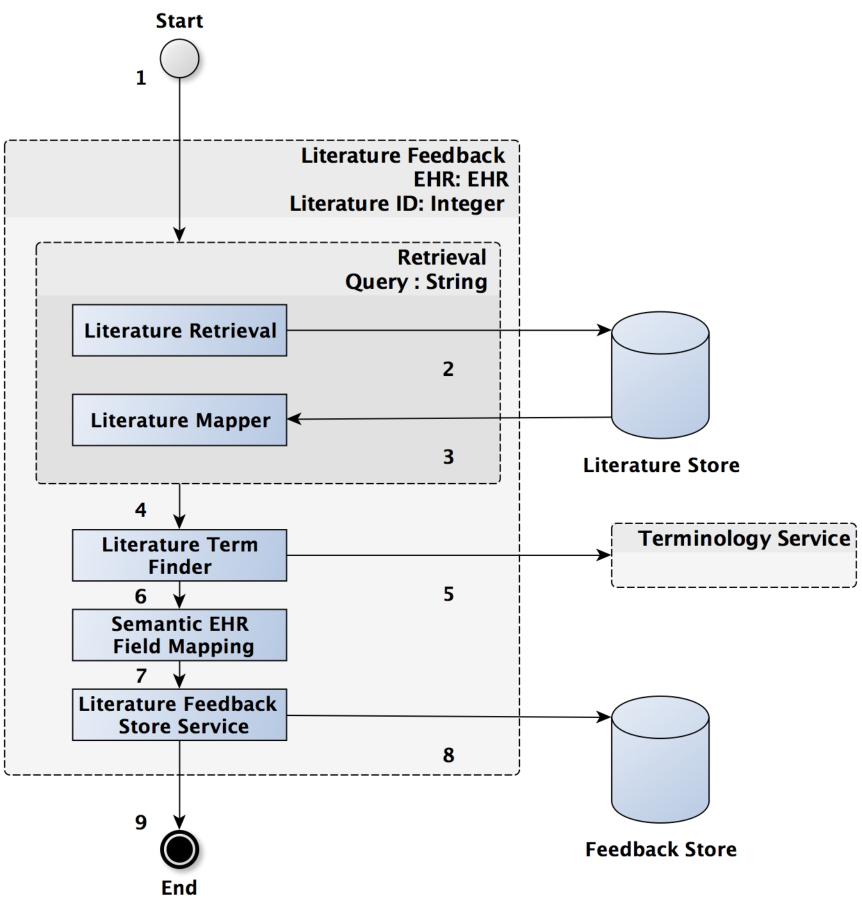


Figure 5.6: UML Activity Diagram: Literature Service - User Feedback

1. On invocation, an EHR and a literature ID is provided to the literature service.
2. A lookup for the literature ID is triggered in the literature store.
3. As soon as the literature store has found the literature, it will be send to the literature mapper, its task being to map the literature to an internal representation.
4. The literature's title and abstract is split into sentences and words. Per sentence words are concatenated to enable term lookup for multi-word terms.
5. Using the terminology service, the (multi-word) sentence parts are searched for known terminology. Duplicate found terms are removed.

6. As for a normal literature search, the EHR is processed by the semantic EHR field mapping component (described in subsection 5.2.1, Semantic EHR Field Mapping), returning a list of ’SearchField’ objects.
7. Both the terms of the literature and the result of the semantic EHR field mapping are given to the literature feedback store service. Its task is to trigger the storage in the feedback store.
8. All information based on the user feedback are persisted with the EHR ID, literature ID and user vote in the feedback store.
9. The process ends. No feedback to the invoker is given because the corresponding view is already updated after triggering the process.

MlInputData Class

As described in step 8 all ’SearchField’ and ’Term’ objects from the EHR and literature are persisted using the feedback store. The feedback store operates on a class used as data container, shown in the UML class diagram of figure 5.7. All class attributes are private and accessed via get and set methods. No methods are shown in the diagram.

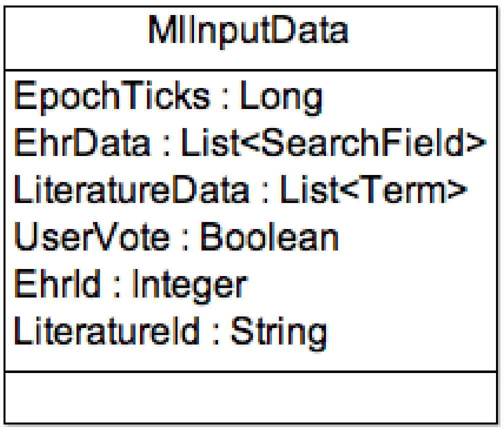


Figure 5.7: UML Class Diagram: MlInputData

The classes ’EpochTicks’ attribute of type long has two purposes, it holds the feedback creation time and serves as object identifier like a primary key does in a relational table. Second class attribute named ’EhrData’ is a list of ’SearchField’ objects, it holds the information extracted from the semantic EHR field mapping, described in subsection 5.2.1, Semantic EHR Field Mapping. Third class attribute with name ’LiteratureData’ being a list of ’Term’ objects holds the terms of the literature. Fourth attribute being of type boolean and named ’UserVote’ indicates if the user performed an up-vote or down-vote. The two attributes ’EhrId’ and ’LiteratureId’ link the user feedback to the corresponding ’Ehr’ and ’Literature’ object, both use the appropriate datatype the objects employs as ID, being ’Integer’ for the EHR and ’String’ for the Literature.

5.3.1 Literature Term Finder

The idea of the literature term finder is to use the terminology prepared in chapter 4, Terminology Service without any context for finding terms in the literature abstract and headline. All terms are the expected outputs for one literature in the machine learning part of the semantic query term identification component, hence, the found terms are ultimately used during literature search.
To accomplish the goal, first the abstract and headline are normalized to a set of sentences. Those are inspected for term occurrences. Part of speech tagging may be used for term identification.
Because the terms from the terminology service incorporate multi-word terms a white space word tokenizer alone is insufficient, only single word terms can be found. An additional step after word tokenization takes place, prior to the term search. Multiple words are concatenated again, to enable the search for multi word terms.
When all terms for one literature are found, duplicate terms are removed. The result consists of a list of terms with unique term labels.

Chapter 6

Implementation

The implementation of all modules both for the terminology service and literature service takes place in C# while the user interfaces parts are programmed in Javascript and HTML. Aside the implementations described in this chapter, an additional C# (test) project is created containing test classes and methods. Almost all of those tests are internally using static inputs and assertions on expected outputs and calculated outputs for the terminology service as well as the literature service. No test is further described in the following sections, as they are used for testing only and do not influence the actual program in any manor with one exception, the terminology services ETL process (which fills the terminology store with data from the ontologies) is triggered by a test method.

6.1 Terminology Service

The service described in chapter 4, Terminology Service, is being implemented, and as previously mentioned it disposes of two operation modes: The online mode, where terminology is proposed according to the context, the users term (part) and the amount of expected suggestions and the offline mode where the terminology for the proposals is loaded from various sources, transforming it on the way and preparing for fast query prior to storing it in the terminology store.

6.1.1 Solr as Term Store

Core of the terminology service is the terminology store, where all terms are deposited. As basis for the store, Apache Solr (details can be found in subsection 3.5, Apache Solr) is chosen. The setup begins by creating a new Solr core producing default configurations, which are changed as follows.

schema.xml

Using the Term class (described in subsection 4.2.3, Term Data Model) as basis, a fitting schema is implemented. In addition to the attributes originating from the Term class the field named 'autocomplete_text' is introduced, used as target during field copy of the 'Label' field and 'Synonyms' field and basis for later queries. The 'Synonyms' and 'Broader' multiplicity (being lists in the Term class) is handled by defining each of the fields with the 'multiValued' attribute set to 'true'. All fields with type enumeration in the Term class are stored as string. Solr's document unique key is mapped to the Term classes 'TermID' attribute. Only the fields 'Label' and 'TermID' are marked as required, all other are optional to enable the storing of incomplete terms e.g. without a description. Base datatype for all fields but 'autocomplete_text' is the type 'solr.TextField' with the attribute 'positionIncrementGap' set to 100 to have values of multivalued fields treated as seperate terms while searching. The 'autocomplete_text' field type is more customized: on query time the query data is split on whitespace and put to lower case and on index time the data is split on whitespace and put to lower case. To enable queries ignoring case as well as looking for parts occurring not in the first word of a multi word term. On index time for each word ngrams are created. The whole schema.xml can be found in appendix D.1, Solr - schema.xml.

solrconfig.xml

The by default created solrconfig.xml file is changed minimally. Only the default search field is configured to be the previously created 'autocomplete_text' field. It eases up manual tests, because the default query endpoint can be used without an additional parameter specifying the search field.

6.1.2 SolrNet Client Library

To insert/update and query terms in Solr from C#, the SolrNet client library is used. The combination of scoring requirements and client library has an impact on the implementation. In detail the requirements are boosting on index time and availability of scores while querying to allow context base result reordering. The following implementation takes place:

1. Two separate Solr documents (one for storing, one for querying) are used.
2. All queries are configured to include the 'score' field in their result.
3. All insert/updates use a certain method to insert documents with their default boost.

The usage of two Solr documents is caused by library method limitations.

Insert and Update

Insert and update are implemented in one method as Solr treats an insert on an already existing 'SolrUniqueKey' as update. An excerpt of the method is shown in listing 6.1, Insert and Update to Solr, on method invocation a list of Term objects is provided and

stored in the 'Terms' variable. The method has no return value.
SolrNet expects the boost value as separate parameter and not as part of the document. To compensate the behavior a 'for each'-loop (line 4 – 6) is used to place all terms into a 'Dictionary' storing the boost as value to the Solr document (line 5). The complete Solr insert document can be found in appendix D.2, Solr - Term Insert Document. Prior to adding the dictionary (line 9), the Solr connection instance is located via the 'ServiceLocater' call (line 8). The Solr connection is initialized inside the static constructor of the class (not shown). As soon as all Solr insert documents are added, the commit to Solr is triggered (line 10), signaling Solr to store all documents persistently.

```
 1 ...
 2 Dictionary<SolrTermInsertDocument, double?> SolrDocumentsWithScore = new
       Dictionary<SolrTermInsertDocument, double?>();
 3   foreach (Term Term in Terms)
 4   {
 5     SolrDocumentsWithScore.Add(SolrUtils.convertFrom(Term),
           SolrUtils.Term.TermSourceScore(Term.Source,
          Term.TermSourceScore(Term.Source,Term.Category));
 6   }
 7
 8 var solr =
       ServiceLocator.Current.GetInstance<ISolrOperations<SolrTermInsertDocument>>();
 9 solr.AddRangeWithBoost(SolrDocumentsWithScore);
10 solr.Commit();
```

Listing 6.1: Insert and Update to Solr

As explained below, the above listing uses in line 5 two utility methods.

ConvertFrom Method

The 'convertFrom' method converts a given 'Term' to a 'SolrTermInsertDocument'. First a new instance of the target object is created, and all values of the source object attributes are copied to the corresponding attributes of the target object, resulting in an 'SolrTermInsertDocument' showing exactly the same values as the 'Term'.

TermSourceScore Method

Purpose of the method is to return the default score to a given category and source ontology as described in subsection 4.2.2, Score-Based Ranking. The whole method is essentially made up by switch case statements. First the given category is evaluated. Second the source ontology is checked. The corresponding ontology score is returned to the invoker. If a not configured configuration occurs (e.g. 'Anatomy' as category and the The Drug Ontology as source) a system exception is raised. Basis for the scores is the ranking of the ontologies found in subsection 4.2.1, Ontology Selection. The result of this method is a double value larger than 1.0.

Query

An excerpt of the query method is shown in listing 6.2, Querying Solr, on method invocation a string containing the (partial) term to query is given and stored in the 'Query' variable. The methods result is a list of SolrTermQueryDocument objects.
By default Solr does not return the document score while querying, however, providing the option displayed in line 6 to SolrNet all Solr fields including the score are returned. Via the 'Rows' parameter the maximum number of rows returned by Solr is set to 1000 (line 7), because results from diverse categories are desired.

```
1 ...
2 List<SolrTermQueryDocument> Results = solr.Query(
3   new SolrQueryByField("autocomplete_text", Query),
4     new QueryOptions
5     {
6        Fields = new[] { "*", "score" },
7        Rows = 1000
8     }
9     ).ToList();
```

Listing 6.2: Querying Solr

The complete query document can be found in appendix D.3, Solr - Term Query Document

Mapping of all Solr schema fields to both Solr documents is implemented by annotations in the documents.
By accessing Solr using the SolrNet C# client library, there is no need for accessing the term store directly from the outside. By means of security the operation mode is comparable to the previously mentioned intelligent proxy access described in subheading 'Solr security' of section 3.5, Apache Solr.

6.1.3 ETL

Each ETL datasource is configured by specifying the loader type with loader specific information and a general information (the category). Two loaders are implemented, a loader for the National Cancer Institute Thesaurus can be implemented any time at a later project stage. Because of the literature service relying on PubMed as literature store, the implementation of a MeSH loader has a higher priority.

1. MeSH loader
   For MeSH the entry point ID and a tree ID exclusion list is provided.

2. Xlsx loader

   Because all ontologies in Xlsx format share the same structure, one loader for all Xlsx formatted ontologies is used. On call, the complete path to the Xlsx file is provided alongside the source name.

Both loaders are explained in more detail, describing the structure of the ontologies along the way.

MeSH Loader

The MeSH download consists of different data files, each describing a certain aspect while some information are redundant. Currently the MeSH loader is invoked for four category fitting subtrees on ETL time being Anatomy (prefix: A; excludes: A, A18, A19, A20, A21), Diseases (prefixes: C, F03; excludes: C, F03, C22, C23, C23.888), Symptoms (prefix: C23.888), and Activity (prefixes: E, F04; excludes E, E07). All terms with an exact MeSH treed ID match in the corresponding exclude list are omitted prior to including all terms matching the prefixes. Details about MeSH can be found in appendix A.11, Medical Subject Headings.

mtrees2015.bin

Contains the MeSH tree of 56341 words and IDs. The tree is a hierarchical structure, each entry consists of a term label and the MeSH tree ID. Example data:
Body Regions;A01
Anatomic Landmarks;A01.111
The complete subtree list can be found at: [51].

desc2015.xml

Consisting of 27456 'DescriptorRecord' entries, each entry has administrative information, e.g., term creation date, an internal ID and further semantic information about the term starting from a MeSH 'TreeNumberList' to a 'ConceptList' containing another term list and other information. In the 'ConceptList' a preferred concept can be found, its 'ScopeNote' describes the term. It is used as term description. All terms in the preferred concept are used as synonyms for the term.

supp2015.xml

Has 686980 'SupplementalRecord' entries. Using the term ID, it is possible to look up the terms indexing frequency and other information. It is not used in the MeSH ETL.

In addition for both XML files a document type description ('.dtd' file) is offered.
Only one instance of the MeSH loader is created and re-used for all subtrees. On constructor invocation of the MeshLoader class two file paths as string are required. The first parameter is the path to the 'desc2015.xml' file and the second is the path to the 'mtrees2015.bin' file. Prior to any processing, the presence of both files is checked. If one of the files is not in place

a fitting exception is raised. After the presence of both files is assured both paths are stored in a class variable

First the 'desc2015.xml' file is processed by a private method. It's program code shown in listing 6.3 - a description is offered after the listing. Essentially all necessary information to create term objects are retrieved in an intermediate term format. The list of intermediate term objects are stored in a member variable of the MeshLoader with private visibility named 'AllMeshTerms'.

As soon as all intermediate term objects are stored, the 'mtrees2015.bin' file is loaded into a C# 'Dictionary' class instance, splitting all lines on semicolon. Each 'Dictionary' entry consists of the tree ID as key and the term label as value. The 'Dictionary' is placed in a private visible member variable of the class named 'MeshTreeIdToName'.

An actual MeSH subtree extraction is triggered by calling the 'getSubtree' method, providing MeSH IDs in the entry point list and the exclude hash set as well as the target category. Taking the (entry point) MeSH ID list and the exclude hash set with MeSH IDs into account, the MeSH terms stored in the 'AllMeshTerms' variable (a list of all terms in intermediate format) are identified.

Code Listing 6.3 shows the extraction of MeSH terms into the intermediate format.

```
private IList<MeshTerm> LoadAllTermsFromXml()
{
  XmlReader textReader = new XmlTextReader(MeshDescXmlPath);
  var query = from XElement c in
      XElement.Load(textReader,LoadOptions.None).Elements("DescriptorRecord")
  select new MeshTerm
  {
    //MeSH Tree IDs
    treeIds = c.Elements("TreeNumberList").Any()
    ?
    c.Element("TreeNumberList").Elements().Select(x =>
        x.Value.Trim()).ToList() : null,

    //ID
    id = c.Element("DescriptorUI").Value,

    //Label
    label =
        c.Element("DescriptorName").Elements("String").First().Value.Trim(),

    //Concept
    concept = c.Elements("ConceptList").Elements("Concept").Where(x =>
        x.Attribute("PreferredConceptYN").Value == "Y" &&
        x.Elements("ScopeNote").Any()).Count()
      > 0 ?
21          c.Elements("ConceptList").Elements("Concept").Where(x =>
                x.Attribute("PreferredConceptYN").Value == "Y").Select(x =>
                x.Elements("ScopeNote").First().Value).ElementAt(0).Trim() : "",
22
23        //Synonyms
24        synonyms = c.Elements("ConceptList").Elements("Concept").Where(x =>
              x.Attribute("PreferredConceptYN").Value == "Y" &&
              x.Elements("TermList").Any()).Count()
25          > 0 ?
26          c.Elements("ConceptList").Elements("Concept").Where(x =>
                x.Attribute("PreferredConceptYN").Value ==
                "Y").Elements("TermList").Elements("Term").Elements("String").Select(x
                => x.Value.Trim()).ToList(): null,
27     };
28
29     return query.ToList();
30  }
```

Listing 6.3: MeSH desc2015.xml - XML Parsing with LINQ

As indicated in line 5 of code listing 6.3 the intermediate format is a private class named 'MeshTerm' with five attributes named treeIds (line 8), id (line 13), label (line 16), concept (line 19) and synonyms (line 24). Because the actual term object has special requirements (as mentioned in 4.2.3, Term Data Model, like each terms broader term is represented by term label and not by ID), an intermediate format is used. In addition the intermediate format eases up maintenance by decoupling the XML mapping from the program code to fulfill the special requirements. The conversion to actual term objects takes place in the 'getSubtree' method by creating term object described as follows.

Term.SourceId

The term source ID is filled with the intermediate term objects attribute named ID. The value is extracted to the intermediate object by the LINQ query shown in line 13 of listing 6.3 and originates from the 'DescriptorUI' XML attribute.

Term.Label

As value for the term label the label from the intermediate term object is used, it's value is extracted from XML by the code shown in line 16.

Term.TermId

Contains almost the same value as the label, only the 'MESH:' prefix is added.

Term.Source

Is hardcoded to 'MeSH' and placed into the term object.

Term.Category

The value is provided on invocation, the data is placed into the term object's attribute, no additional processing of the value takes place.

Term.Definition

Basis for the term definition is the concept attribute of the intermediate term object. The string value originates from XML and is extracted by the code shown in lines 19 - 21 . Because in the XML file the concept is an optional value, the LINQ query used for extraction is more complex, prior to extraction in line 21 the attributes presence is asserted in lines 19 and 20. In case no concept is found for the term, the XML parsing returns an empty string (line 21).

Term.Synonyms

All synonyms are retrieved from the intermediate term format. Basis for all synonyms is the 'synonyms' attribute of the intermediate term object. In the MeSH XML file the corresponding attribute is optional, hence the extraction of the individual synonym is more complex as shown in lines 24 - 26 of code listing 6.3. If no synonyms are found by the XML parsing, null is returned. If synonyms are available in the intermedia format, the list of synonyms is transformed to a hash-set ignoring case.

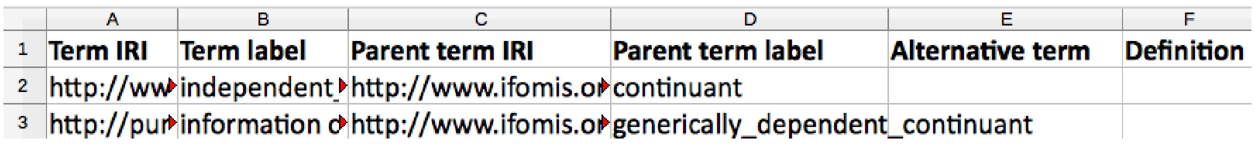
Term.Broaders

Broader terms are found by removing all numbers from the end until the most right full stop of all MeSH tree IDs available in the intermedia term object and looking up the corresponding label via the dictionary stored in the MeshTreeIdToName variable. The actual extraction from XML is shown in lines 8 - 10

A MeSH subtree loading ends by returning the list of terms.

Xlsx Loader

All ontologies retrieved from Ontobee [52] share the same Xlsx column names and structure. An example of the column names using the DrOn ontology (details about DrOn can be found in appendix A.1, The Drug Ontology) is shown in the screenshot 6.1 below.



	A	B	C	D	E	F
1	Term IRI	Term label	Parent term IRI	Parent term label	Alternative term	Definition
2	http://ww	independent_	http://www.ifomis.o	continuant		
3	http://pur	information o	http://www.ifomis.o	generically_dependent_continuant		

Figure 6.1: Ontologies in Xlsx Format - Column Names

The Xlsx loader is made up by different private methods which are called inside the public 'loadFile' method returning a list of Term class instances for a given term source name, category, absolute path and name of a Xlsx file. In the following, all (private) methods involved are described.

GetConnectionString

A connection string for the given Xlsx file is created by using properties, indicating the Xlsx format, path and filename.

ReadExcelFile

Given the connection string the Xlsx file is loaded and all cells are selected using an 'OleDbCommand' in transact SQL style in a loop over all sheets. The transact SQL style command used for cell retrieval is shown in listing 6.4.

```
OleDbCommand.CommandText = "SELECT * FROM [" + SheetName + "]";
```

Listing 6.4: OleDbCommand for Column Selection

All cells returned from the 'OleDbCommand' invocation by the 'OleDbDataAdapter ' are stored in an C# 'DataTable' per sheet. To return all 'DataTable's for further processing, they are stored in a 'DataSet' class instance.

ConvertToTermList

On invocation the 'DataSet' and term source name and category are provided. The mapping of each column to the corresponding term class attribute takes place by accessing the cells of the 'DataTable' via index in a 'for each'-loop over all tables in the 'DataSet'. Primitive term cleaning takes place per term, in case the term label contains a square bracket, the square bracket alongside its content is removed. The result is a list of Term objects.

ConvertBroaderIDsToLabels

All terms in the Xlsx formatted ontologies use IDs to reference broader terms and not their actual label. In order to replace the IDs with the corresponding label, first all terms are placed into a dictionary using the term ID as key and the corresponding label as value. Applying two nested loops on the term list, for each term all synonym IDs are looked up in the dictionary and each broader term ID is translated to its actual term label.

A Xlsx loader ends by returning the list of terms.
By modifying the 'OleDbCommand' in listing 6.4, OleDbCommand for Column Selection, to fit another column order or by skipping columns, the Xlsx loader can be easily adapted for future ontologies because the convertToTermList method operates on indexes only. In addition the use of private methods enables easy skipping or adding steps of the Xlsx loader.

Global Term Label Exclude List

A global exclude list is applied (to all terms from the MeSH and Xlsx loaders) by using string comparison with ignore case option on each 'Term' objects label and each value in the exclude list. As the label is not part of the exclude list it is placed in the result list. Values in the global exclude list are: review of, associated with, severe, rate, time, human.

Saving Terms to Solr

All via the global exclude list filtered Term objects are saved to a file in binary format. By saving the terms to a file, the extraction and transformation part of the ETL is separated from the loading part. In a future step the ETL may be divided into two separate processes. Accessing the previously stored terms in the file, all terms are saved in the terms store using the 'insert and update' method described in subsection 6.1.2, SolrNet Client Library. Aside from saving the Term object in Solr, they are provided to the ETL invoker once at the end of the ETL.

6.1.4 Semantic Result Optimization

All dimensions, but the "match count" and the "user's preference", mentioned in subsection 4.3.1, Semantic Result Optimization, are implemented. Both not implemented dimensions may be added at a later project stage. Details about the implementation are mentioned below.

Context Specific

1. To evaluate the match position of the search term either at the beginning of the term label or as part of a term label, regular expressions are used. An example of the implementation can be seen in line 8 and 15 of listing 6.5, Excerpt of the Adjust Score Method.
2. The category match and number of free slots in the result set for a certain category is taken into account by using a proportional assignment based on the provided category weight and number of free slots in the result set. A complete description can be found in the current section's subheading 'Result Reordering by Categories'.

Term Static

1. By counting the spaces in each individual term in the query result list, a variant of the 'term length' is implemented as seen in line 7 and 14 of listing 6.5, Excerpt of the Adjust Score Method.

2. While storing the terms in the term store a base boost is given per term, implementing a 'category default boost'. The implementation is shown in section 6.1, Insert and Update to Solr.

Aside the category boosting (term static dimension 2), taking place while storing terms in the term store, two methods implement the query time optimizations. Each of them is described below.

Score Adjustment by Term Label and Search Term

The following program code (listing 6.5) shows the 'adjustScore' method, implementing the context specific dimension 1 (line 8 and 15) and term static dimension 1 (line 7 and 14). Both dimensions are evaluated together.

```
 1 public static SolrTermQueryDocument adjustScore(SolrTermQueryDocument
     SolrDocument, string SearchTerm)
 2 {
 3   ...
 4   int SpacesCount = Suggestion.Count(Char.IsWhiteSpace);
 5
 6   // case starting match & single word => Score * 1.0
 7   if (SpacesCount == 0 &&
 8       Regex.Matches(Suggestion, "(?i:.*^" + SearchTerm + ".*)").Count > 0)
 9   {
10     Score *= 1.0;
11   }
12   ...
13   // case match & more than two words => Score * 0.5
14   else if (SpacesCount > 2 &&
15       Regex.Matches(Suggestion, "(?i:.*" + SearchTerm + ".*)").Count > 0)
16   {
17     Score *= 0.5;
18   }
19   ...
20   SolrDocument.Score = Score;
21
22   return SolrDocument;
23 }
```

Listing 6.5: Excerpt of the Adjust Score Method

Result Reordering by Categories

The statements implementing the reordering (context specific dimension 2) are placed directly in the terminologies entry point method called 'autosuggest', because its the last step prior to term delivery.

For category weight (context) based result reordering the term amount per category in the result set is calculated. The basic idea is a proportional assignment. First all category weights are added. Dividing the amount of expected terms by the sum of weights, the term quotient is calculated. By multiplying the term quotient and the weight of each category, the term count per category is decided. All categories with their term count are saved in a list and sorted in descending order by the term count.
As soon as the duplicate free, score adjusted and re-ordered query result arrives via the SolrNet client from the term store, the term count per category is used to extract terms for each category. Depending on the chosen result ordering type, the terms per category are placed in the result set. Two ordering types are implemented. 'Category Diversity' is the default ordering type.

Category Diversity
Alternating between the categories, starting with the highest weight, one term per category is added to the result list. As the slots for one category are depleted, the category is skipped.

Category Clusters
Starting with the highest weighted category, all terms per category are added to the result list.

A side affect of using the product of quotient and category weight (producing a float value) as slot counter for one category in the result set (expecting an integer) is the rounding error. It is solved by facilitating C#'s 'Math.Floor' method prior to the integer cast. However, for certain combinations of category weights and expected term count, the usage of 'Math.Floor' leads to less terms than expected.

6.1.5 Duplicate Term Removal

Duplicate terms are removed both on query time and ETL time using the method shown in listing 6.6, Duplicate Term Removal.
The duplicate removal on query time takes place, because terms could be added to the term store while the program is running, resulting in duplicates not known on ETL time.
Basic idea of the method is, as described in 4.2.4, Duplicate Term Removal, to compare one term against all other terms by term label, taking the synonyms into account and if a duplicate is found the base boost and the term completeness is consulted to decide which term to favor.
On method invocation a list of term objects is provided, stored in the 'ListWithDuplicates' variable, possibly containing term with duplicate labels as well as a boolean variable indicating wether or not duplicates should be handled ignoring case (line 1). The ignore case option

is set to 'true' by default. In line 5 the dictionary named 'TermDictionary' with string as key and term object as value is declared and initialized. The instantiation of the dictionary depends on the state of the ignore case option, either the dictionary is handling its keys ignoring case (line 8) or not (line 12). Line 14 shows the 'for each'-loop used to iterate the given term list. The loops body is starting in line 15. As indicated by the comment in line 16, in line 17 the term label is checked wether or not it is already present in the dictionary using a fitting if statement.
If the term label is present, the corresponding term is retrieved from the dictionary and stored in the variable named FoundTerm via the out parameter modifier of the TryGetValue method invocation on the dictionary. In line 20 the rating of the found terms is compared to the rating of the current term. If the rating of the FoundTerm is smaller than the rating of the current term, the value of the dictionary at place of the FoundTerm label is changed to the current term in line 21.
As indicated by the comment in line 24 the 'else if' in line 25 checks wether or not the current term has any synonyms. All statements from line 25 till line 49 take place only if the term label is not already present in the dictionary and if the term has synonyms. In line 26 the SynonymInDictionary boolean variable is set to false. Using a 'for-each' loop starting in line 29 over all synonyms of the term, the presence of the synonym in the dictionary is determined. Similar to checking the term label presence and replacing the term object in the dictionary if the current term has a higher rating, a check for each synonym and depending on the rating the term in the dictionary is replaced. The program code in lines 30 to 37 is identical to the one in lines 17 to 22 with one exception, in line 32 the boolean variable named SynonymInDictionary is set to true, to indicate that at least one synonym of the term is in the dictionary. After iterating all synonyms checking their presence in the dictionary and replacing terms if the rating of the new term is higher, in line 41 it is determined if at least one synonym of the current term is present in the dictionary by evaluating the SynonymInDictionary using a fitting if statement. In case no synonym of the current term is found in the dictionary, the term label and all its synonyms are added to the dictionary, each time using the term as value (lines 43 to 47).
As indicated by the comment in line 50, the lines 52 till 54 are only executed as the term label is not present in the dictionary and the term has no synonyms. In line 53 the current term label used as key and the term object used as value is added to the dictionary.
In line 56 a hash-set of type term is created. Facilitating a 'for-each' loop (line 57) over all values of the dictionary a term object is added to the hash-set if it is not yet included. In line 59 the if-statement is shown to determine the presence of the term-object in the hash-set. The hash-set's ToList method is invoked and the resulting list of term objects is used as return value in line 61.

```
public static IList<Term> removeDuplicats(IList<Term> TermList, bool
    IgnoreCase =true )
{
  Term FoundTerm;
  bool SynonymInDictionary;
  Dictionary<string, Term> TermDict;
  if (IgnoreCase)
  {
    TermDict = new Dictionary<string,
        Term>(StringComparer.InvariantCultureIgnoreCase);
  }
  else
  {
    TermDict = new Dictionary<string, Term>();
  }
  foreach(Term Term in TermList)
  {
    //case term label is known
    if (TermDict.TryGetValue(Term.Label, out FoundTerm))
    {
      //case new term has higher rating than known term, replace
      if (FoundTerm.Rating() < Term.Rating())
        TermDict[FoundTerm.Label] = Term;
    }
    //case term label is not known but term has synonyms
    else if (Term.Synonyms != null)
    {
      SynonymInDictionary = false;
      //check the presence of the synonyms
      foreach (string synonym in Term.Synonyms)
      {
        if (TermDict.TryGetValue(synonym, out FoundTerm))
        {
          SynonymInDictionary=true;
          //case new terms synonym has higher rating than known term, replace
          if (FoundTerm.Rating() < Term.Rating())
          {
            TermDict[FoundTerm.Label] = Term;
          }
        }
      }
      //case neither term label and synonym is known, add term label and
          synonyms
      if(!SynonymInDictionary)
      {
        TermDict.Add(Term.Label, Term);
        foreach (string synonym in Term.Synonyms)
        {
          TermDict.Add(synonym, Term);
        }
      }
    }
    //case term label is not known and has no synonyms, add term label
    else
    {
      TermDict.Add(Term.Label, Term);
    }
  }
  HashSet<Term> TermHashSet = new HashSet<Term>();
  foreach(Term Term in TermDict.Values)
  {
    if (!TermHashSet.Contains(Term)) TermHashSet.Add(Term);
  }
return TermHashSet.ToList();
}
```

Listing 6.6: Duplicate Term Removal

Term Rating Method

This static method with public visibility returns a double value which is calculated as follows: The base value is decided by evaluating the term objects category and ontology (described in 4.2.2, Score-Based Ranking). Than the presence of the term objects synonyms, description and broader terms is determined. Each time one of the attributes is present, 0.1 is added to the base value.

6.1.6 Highlights of the Term View Component

Because the terminology service can be used for various input fields, and to ease maintenance, the whole front-end part written in HTML and Javascript consisting of the autocomplete function as well as utility methods is placed into an own file and imported into the central template for the HTML head part of the webpage. The signature of the autocomplete function can be seen in listing 6.7. The first parameter is the interface component where the terminology service is attached to (e.g. a HTML text area or input field element). Second parameter is the maximum amount of suggestions to be retrieved. All six remaining parameters specify the individual weight per category.

```
function autocomplete(ViewElement, ElementCount,
                MedicationWeight, ActivityWeight,
                DiseaseWeight, GeneWeight,
                SymptomWeight, AnatomyWeight) {...}
```

Listing 6.7: Function Signature of the Terminology View Component

The autocomplete function relies on JQuery to locate the HTML element for attachment. All user inputs arriving in the HTML element are retrieved by a utility function which extracts all words after the last ',' character, by splitting the input field on ',' and returning the last element of the split. Using the JQuery 'autocomplete' feature a HTTP POST request is formulated and sent to the back-end, containing the (partial) user input, maximum result count and the weights per category. The response of the back-end in JSON format is transformed to match JQueries autocomplete feature. Because of the category icons, highlighting and synonyms the '_renderItem' function is overloaded. Each of the three features are explained as follows.

Category Icon

By taking the category attribute from the back-end result into account, the URL of the category icon is created.

Highlighting

Per term result and corresponding request the result term is checked for the search term and highlighting takes place. In case the result term does not contain the search term, the synonyms are checked and if a match is found, it is highlighted. All highlighting is implemented with regular expressions. An example can be seen in listing 6.8. All matches ignoring case (line 2) are replaced with the actual match surrounded by HTML tags containing a CSS class used for highlighting (line 3). In using the actual match and the ignore case option, the case of the original word remains unchanged.

```
1 var newText = String(item.value).replace(
2   new RegExp(lastTerm, "gi"),
3   "<span class='ui-state-highlight'>$&</span>");
```

Listing 6.8: Term Highlighting

Synonyms

By using an ignore case match on the user input to the search terms found, synonyms are only displayed as the term itself does not match the searched term. In addition, the first matching synonym is displayed only.

To display tooltip information, the injection point 'focus' of JQueries 'autocomplete' is used. Basis for the tooltip information is the literature object's 'additionalInformation' attribute.

6.2 Literature Service

As described in chapter 5, Literature Service, the literature service retrieves useful and EHR context fitting information displayed by special interface components to the user. On service call an EHR is provided and by using different modules NLM's PubMed literature store is queried to retrieve literature.

6.2.1 Semantic EHR Field Mapping

As described in subsection 5.2.1, Semantic EHR Field Mapping, the mapper creates a list of 'SearchField' objects from the EHR provided on invocation. Fifteen out of appoximately fifty EHR fields are mapped to twelve EHR field categories in using the semantic field mapper. Each of the fifteen fields is individualey mapped, producing one or several 'SearchField' objects. All mapping takes place in an utility class. Two mapping examples are stated below.

Patient Age

In the applications EHR edit mode a date chooser is offered to the user from which the patients date of birth (DOB) can be chosen. The DOB is mapped to words describing the age by the program code shown in listing 6.9. Basis for age filtering is a table found on [53]. In line 3 the patients DOB is retrieved from the EHR. All following steps take places only in case the DOB from the EHR contains a value (line 4).

Basis for all age calculations is the current date (line 6). In line 7 the result string ('AgeResultString') is instantiated and initialized. For each word the individual date is calculated by adding a negative value using the appropriate method (examples in line 9 and 12) except for "80 and over" because its at the top of the age scale and used as default value as all other dates do not fit. As shown in line 15 and 20, the patients DOB is compared in age ascending order facilitating 'else if' control flow statements. All comparisons use the 'CompareTo' method of C#'s 'DateTime' data type, returning a negative value if the value on the left hand side shows an earlier date. The 'AgeResultString' variable is set to the corresponding words describing the patients age (line 17, 22, 26) prior to adding a 'SearchField', it is created by the 'SearchFieldFactory' providing name, data type and value. In line 28 the search field is created and added to the list of mapped EHR fields named 'SearchFields'.

```
1 ...
2 FieldName = EhrField.AGE;
3 DateTime DateOfBirth = Patient.DateOfBirth;
4 if (DateOfBirth != null)
5 {
6   DateTime CurrentDate = DateTime.Now;
7   string AgeResultString = "";
8   //Newborn: birth-1 month
```

```
  DateTime NewBorn = CurrentDate.AddMonths(-1);
  ...
  //Aged: 65-79 years
  DateTime Aged = CurrentDate.AddYears(-79);
  //80 and over: 80+ years
  ...
  if (NewBorn.CompareTo(DateOfBirth) < 0)
  {
    AgeResultString = "Newborn";
  }
  ...
  else if (Aged.CompareTo(DateOfBirth) < 0)
  {
    AgeResultString = "Aged";
  }
  else
  {
    AgeResultString = "80 and over";
  }
  SearchFields.Add(SearchFieldFactory.create(FieldName,
      LiteratureSearchDataType.STRING, AgeResultString));
}
...
```

Listing 6.9: Excerpt Semantic EHR Field Mapping - Patient Age

Other Medication

The EHR contains a field where other medications the patient recieves are entered. While using the program code shown in listing 6.10, the EHR field is transformed into search field object for further processing. Field data is in string format and can contain more than one medication. One medication can consist of more than one word. All medications in the field are separated by commas on user input, facilitating the terminology service. After setting the field category using the 'EhrField' enumeration to type 'MEDICATION' (line 2), the field from the EHR is saved in a local variable (line 3), which is analyzed as follows.

Caused by line 4, the following takes place only if the medication field contains data. Should the medication string contain a comma (line 6), indicating the presence of more than one medication in the field, it is splitted by commas to handle each medication separately. When sperating the string by commas, the option 'RemoveEmptyEntries' is set on invocation of the split method. Using this option, the split method does not produce empty entries (line 8). However, a space after a comma is still possible, hence, in line 10 it is ensured that an actual medication is in the current string. In line 11, for each medication by using the 'SearchFieldFactory' a new 'SearchField' with the name, datatype and value is created and added to the 'SearchFields' result list.

If the medication string does not contain a comma, only one medication is in the EHR field. Prior to adding the medication to the 'SearchFields' result list, a search field is created using the 'SearchFieldFactory' in line 16. On factory invocation the name, datatype and value is provided.

```
 1 ...
 2 FieldName = EhrField.MEDICATION;
 3 string OtherDrugs = Patient.OtherDrugs;
 4 if (!string.IsNullOrWhiteSpace(OtherDrugs))
 5 {
 6   if (OtherDrugs.Contains(","))
 7   {
 8     foreach (string Drug in OtherDrugs.Split(',',
           StringSplitOptions.RemoveEmptyEntries))
 9     {
10       if (!string.IsNullOrWhiteSpace(Drug))
11       SearchFields.Add(SearchFieldFactory.create(FieldName,
             LiteratureSearchDataType.STRING, Drug));
12     }
13   }
14   else
15   {
16     SearchFields.Add(SearchFieldFactory.create(FieldName,
           LiteratureSearchDataType.STRING, OtherDrugs));
17   }
18 }
19 ...
```

Listing 6.10: Semantic EHR Field Mapping - Other Medication

6.2.2 Rule Based Semantic Query Term Identification

The list of 'SearchFields' objects created during mapping of the EHR (described in subsection 6.2.1, Semantic EHR Field Mapping) is the input data for the rule based semantic query term identification. Each rule creates query terms and a list of words to highlight in the result as described in 5.2.2, Semantic Query Term Identification. Facilitating a 'for each'-loop on the set of 'SearchFields', first, the field exclude hash set is consulted and if the field is not excluded a fitting rule is applied. The exclude hash set contains seven EHR field types. Two prominent rules are shown and explained in the following.

Medication

On invocation the 'SearchField' is handed to the medication rule method (line 1) shown in listing 6.11. Employing the 'getValue' utility method the value of the 'SearchField' is saved in the local variable named 'Value' of type string (line 3). The functionality of the getValue method is described after the listing. Using the 'string.Compare' method

with ignore case option the 'Value' is compared to the string 'immunotherapy' in line 4. If a match occurs the 'Value' is replaced by the narrower string 'Ipilimumab' in line 6. Near the methods end in line 9 a new 'RuleResult' object is created.
The first parameter of the 'RuleResult' constructor is the search string. Content of the search string is the content of the 'Value' variable concatenated with ' efficacy' interconnected with 'OR' to another occurrence of the 'Value' variables content concatenated with ' safety'. Each part of the 'OR' as well as the complete search string is surrounded by brackets.
Second constructor parameter is the terms to highlight in the search result separated by a comma (line 11). The content is almost identical to the search string with exception of three differences: No brackets are used, the OR connection is omitted and the 'Value' variables content is added without any concatenation.

```
 1 public static RuleResult medication(SearchField SearchField)
 2 {
 3   string Value = getValue(SearchField);
 4   if (string.Compare(Value, "immunotherapy",true) == 0)
 5   {
 6     Value = "Ipilimumab";
 7   }
 8
 9 return new RuleResult(
10 "((" + Value + " efficacy) OR (" + Value + " safety))",
11 Value + " efficacy," + Value + " safety," + Value);
12 }
```

Listing 6.11: Rule Based Semantic Query Term Identification - Medication

GetValue Method

The 'getValue' method handles the different value types of the 'SearchField' by casting its attribute named 'Value' of type 'Object' to the actual datatype indicated by the attribute named 'Type' of the enumeration type 'LiteratureSearchDataType'. Because the methods return value type depends on the context of the invocation, ultimately on the provided 'SearchField', the return value is of type 'dynamic'. The type is chosen at execution time.

Issue Type

Identical to the medication rule method, on invocation of the issue type rule method the 'SearchField' is provided as shown in line 1 of listing 6.12. In line 3 the 'SearchFor' variable of type string is instantiated and initialized with 'melanoma ' concatenated with the content of the 'NoMeSHExpension' variable. By using the 'NoMeSHExpension' the PubMed search engine is configured to not extend the word by narrower terms using MeSH. The whole content of the 'SearchFor' variable is surrounded by brackets.

Line 4 the 'MarkInResult' variable is instantiated as type string and initialized with 'melanoma'. By using the getValue method in line 5 the 'SearchField' 'Value' attribute is retrieved and saved in a local variable of type string named 'Value'. The 'Value' variable is checked by string comparison with ignore case option wether or not its content equals 'metastatic melanoma' (line 6). Should the check be successful, the 'SearchFor' variable is appended by 'AND (metastatic)' in line 8 and the 'MarkInResult' variable is concatenated with ',metastatic'(line 9). The idea of the rule is to implement logic for all issue types as above. At the methods ending in line 13 a 'RuleResult' object is created, handing over the local variables 'SearchFor' and 'MarkInResult' to the constructor.

```
 1 public static RuleResult issueType(SearchField SearchField)
 2 {
 3   string SearchFor = ("(melanoma " + NoMeSHExpension + ")");
 4   string MarkInResult = ("melanoma");
 5   string Value = getValue(SearchField);
 6   if (string.Compare(Value,"metastatic melanoma", true) == 0)
 7   {
 8     SearchFor += "AND (metastatic)";
 9     MarkInResult += ",metastatic";
10   }
11   else if ...
12 
13   return new RuleResult(SearchFor,MarkInResult);
14 }
```

Listing 6.12: Rule Based Semantic Query Term Identification - Issue Type

6.2.3 Machine Learning Based Semantic Query Term Identification

To enable the search term prediction method relying on learned data for an unknown EHR two additional methods are implemented.

1. Learn Data Collection Method: Based on user up and down votes for certain literature in context with an EHR, learning data is collected and stored in the 'Feedback Store' as described in 5.3, User feedback.
   On method invocation the EHR object and the literature ID are provided. After literature object lookup by ID via the query 'PubMedApiClient' class described in 6.2.5, Quering PubMed, terms are retrieved for both objects, the EHR and the literature, making up the learn data. EHR terms are retrieved via the semantic field mapper (input) and keywords of the voted literature (expected output) are retrieved from two sources, only expected output terms known to the terminology service are used. The two output term sources are

(A) The literature's NLM webpage. A detailed description of the process alongside a code listing is found in 'Literature Term Retrieval from NLM's Webpage'.

(B) Searching the literature title and abstract for terms known to the term store. A description and a part of the extraction method can be found in a subsection of the current subchapter named 'Searching for Known Terms'.

2. Train Method, taking the learn data based on user feedback from the 'Feedback Store' into account the machine learning mechanism, a Multi-Label (ML) Support Vector Machine (SVM) using a linear kernel, is trained. Prior to training, data from the 'Feedback Store' is transformed as shown in subsection 'Preparing Data for Machine Learning'. After training the ML SVM, it is saved alongside the list of used (EHR and Literature) terms to hard disk, facilitating a special class, prepared for a binary serialization. To enable an external invocation of the learning method, the controller used to communicate with the user interfaces offers a method to trigger the learning via a HTTP get on a specific URL.

Literature Term Retrieval from NLM's Webpage

Terms are retrieved by downloading the whole webpage's Domain Object Model (DOM) from NLM per literature, cleaning it from unnecessary information in order that XPath queries can be used to extract terms from three different HTML elements, 'MeshTerm', 'Substance','Keyword' as well as the literature title. A list of all queries used is shown in table 6.1 alongside demo results.

HTML Element	XPath Query	Example Result
MeshTerm	//*[@alsec="mesh"]/text()	Dacarbazine/adverse effects...
Substance	//*[@alsec="subs"]/text()	Dacarbazine...
Keyword	//*[@name="keywords"]/@content	Aged;Antibodies, Monoclonal
Title	//*/h1/text()	Five-year survival rates for treatment-naive patients...

Table 6.1: Literature Term XPath Queries

The functionality is implemented in a class named 'PubMedWebClient', offering one public method to retrieve all terms in an object of class 'LiteratureFurtherInformation' by providing a Literature ID as string on invocation.
Listing 6.13 shows the code used for downloading and term retrieval from the 'Keyword' HTML element, it is the class' most complex extraction. For all tasks the C# 'HtmlAgility-Pack' library is used. After method invocation providing the literature ID (line 1) in a string variable named 'LiteratureId' the 'HtmlWeb' object is instantiated and initialized in line 3. The 'HtmlWeb' class is part of the library and offers a 'Load' method, that downloads a webpage from a given string containing an URL and cleans the download to enable XPath

evaluation. Using the private class readonly string variable named 'SearchTermUrl' containing the base literature URL at NLM, concatenated with the 'LiteratureId' as parameter to the 'Load' method, the method is invoked returning a 'HtmlDocument' instance ready for XPath evaluation in line 4. In line 5 the target list for the terms is created. It is composed of type string 'IList' filled with an empty 'List' of strings. The XPath query evaluation takes place in line 6,7 by calling the 'SelectNodes' method providing the XPath query string on the 'HtmlDocument' object. Result of the XPath query evaluation is an 'HtmlNodeCollection'. After checking the 'HtmlNodeCollection' to be different from 'null' in line 8, a 'for each'-loop in line 10 is employed to iterate over all 'HtmlNode' objects of the 'HtmlNodeCollection'. In the loop's body the 'HtmlNode' text is saved in a local string variable named 'InnerText' by accessing the HtmlNode's 'InnerText' attribute (line 12). As indicated in the comment in line 13, the 'InnerText' variable is checked to not include the string "PubMed" using the 'IndexOf' method with ignore case option and checking if the returned index equals -1, indicating the string's absence in line 14. In case the string does not contain "PubMed", the 'InnerText' variable is split by comma with 'RemoveEmptyEntries' option the resulting string array is saved in a local variable named 'SplittedString' (line 16) which is iterated employing a 'for each'-loop in line 17. In the loop's body (line 18 – 21) each element is first checked to be neither null nor an empty string (line 19), if the check is positive the string is added to the Keywords list in line 20.

A similar processing takes place for all two remaining term information sources found on the webpage as well as the document title (the document title originating from the webpage is used for debugging only). In line 27 a 'LiteratureFurtherInformation' object is instantiated providing all retrieved term lists, the literature title as well as the literature ID (provided on method call) to the constructor. The 'LiteratureFurtherInformation' object is returned to the invoker.

```
1  private LiteratureFurtherInformation getDocumentById(string LiteratureId)
2  {
3  HtmlWeb HtmlWeb = new HtmlWeb();
4  HtmlDocument HtmlDocument = HtmlWeb.Load(SearchTermUrl + LiteratureId);
5  IList<string> Keywords = new List<String>();
6  HtmlNodeCollection HtmlNodeCollection = HtmlDocument.DocumentNode.
7    SelectNodes(PubMedXmlParserConfig.Expressions[WebXmlEntryEnum.KEYWORD]);
8  if (HtmlNodeCollection != null)
9  {
10   foreach (HtmlNode HtmlNode in HtmlNodeCollection)
11   {
12     string InnerText = HtmlNode.InnerText;
13       //sometimes keywords are a list of ncbi databases
14       if (InnerText.IndexOf("PubMed", StringComparison.OrdinalIgnoreCase
           )==-1 )
15       {
        String[] SplittedString = InnerText.Split( new string []{","}, StringSplitOptions.RemoveEmptyEntries);
        foreach (string String in SplittedString)
        {
          if (!string.IsNullOrEmpty(String))
            Keywords.Add(String);
        }
      }
    }
  }
}
...
return new LiteratureFurtherInformation(LiteratureId, LiteratureTitle, MeshTerms, Keywords, Substances);
```

Listing 6.13: Literature Term Retrieval from NLM's Webpage

Literature Term Finder

As described in subsection 5.3.1, Literature Term Finder, each literature objects abstract, title as well as terms retrieved from the NLM webpage are searched for terms known to the terminology service. At a later project stage a part of speech (POS) tagging may be added enabling even more sophisticated term identification. Because the terminology service already offers a normed vocabulary, an extra step to do POS is not needed and could be even error prone.
The functionality is implemented in a class named 'EnrichLiteratureService' offering public static methods to perform the term search on different object types (literature class, list of strings and string). Each public method is internally transforming the object provided on invocation and either calls other public class method or directly the private worker method as well as calling the duplicate removal method (the duplicate removal method is explained in 6.1.5, Duplicate Term Removal). All methods return a list of 'Term' objects.
The private worker method requires a string parameter and first checks, wether or not the given string is the null value or an empty string.
In case the check is positive, an empty term list is returned.
In case the check is negative it is decided if the string ends with a question mark, exclamation mark or full stop. In case one of the characters is found, the very last character is removed. The remaining characters in the string are splitted on white spaces, resulting in a string array which is transformed into a string list. Using the source code shown in listing 6.14, terms are extracted known to the terminology service. Following the listing, each of the three invoked utility methods are explained. Most variable initialization takes place prior to the shown listing. All statements between line 3 and 17 are performed as long as the 'SentenceWordList' contains at least one string, by employing a while loop (line 2). First the

target variable of type Term named ’CurrentTerm’ is assigned with the ’null’ value (line 4). In line 5, a for-loop is used for iteration on the ’SentenceWordList’. The loop’s counter variable is instantiated as type integer and assigned with 1 as value, because the variable is handed over to the ’stringFromList’ method. Line 7 shows the invocation of the ’stringFromList’ method, parameters are 0 as start index, the ’SentenceWordList’ and the current counter variable as end index. The the resulting string is stored in the ’SearchFor’ variable. Which is used in line 8 as input to the terminology service query. The resulting list of terms, is placed in the ’FoundTerms’ variable. Both variables filled with results of the previous method calls (’FoundTerms’, ’SearchFor’) are used as parameter on invocation of the ’find’ method in line 9. The invocations result is stored in the ’Term’ variable. If the retrieved term is different from ’null’ the ’CurrentTerm’ is assigned with the ’Term’ variables value (line 10). Ensuring that the longest term of the current ’SentenceWordList’ state, known to the terminology service is found. After the for-loops end in line 12 the ’CurrentTerm’ variable is checked to be different form ’null’, if the check is positive, the ’CurrentTerm’ is added to the ’ReturnValue’ list, a prior initialized list of term objects used as the methods return value. In line 16, the last statement of the while loop’s body, the first element of the ’SentenceWordList’ is removed. After both loops ended, the ’ReturnValue’ list is returned to the invoker (not shown in the code listing).

```
 1 ...
 2 while (SentenceWordList.Count > 0)
 3 {
 4   string CurrentTerm = null;
 5   for (int i = 1; i < SentenceWordList.Count + 1; i++)
 6   {
 7     SearchFor = stringFromList(0, SentenceWordList, i);
 8     FoundTerms = TerminologyService.queryAllCategories(SearchFor);
 9     Term = find(SearchFor, FoundTerms);
10     if (Term != null) CurrentTerm = Term;
11   }
12   if (CurrentTerm != null)
13   {
14     ReturnValue.Add(CurrentTerm);
15   }
16   SentenceWordList.RemoveAt(0);
17 }
18 ...
```

Listing 6.14: Literature Term Finder

stringFromList

The methods task is the creation of a string, given the start and the end index of a string list, by concatenating all elements in range of the indices using a blank. On invocation the method receives 3 parameters. The first parameter is of type integer, the start index. The second parameter is the string list and the third value is the end index parameter of type integer, it is required to be larger than the start index, because the method internaly employes a for loop, beginning with the start index incrementing its counter variable till it is smaller than the end index (parmater). Result of the method is a string. If one of the index parameters are out of bound or the given list is null, an exception is raised.

queryAllCategories

This is a distinct terminology service method. It queries the terminology store while skipping the Semantic Result Optimization but employing duplicate term removal (described in subsection 6.1.5, Duplicate Term Removal).

find

The task accomplished by the ’find’ method is, matching each individual term in a list of terms by label and synonyms (both being attributes of the term class) to a given string using string comparison and favoring the first matching term object. On method call, the string to look in and the list of terms to search for is provided. The methods return value is a term object. If no match is found, null is returned.

Feedback Store

The feedback store operates on ’MlInputData’ objects (described in the Literature Service chapter) and is implemented by a static class named ’LiteratureFeedbackStore’, offering (static) methods comparable to methods offered by a persistence layer’s Data Access Object (DAO). No update method is implemented because it is not required. Instead of persisting objects to a database, the objects are saved in binary format (employing C#’s ’BinaryFormatter’ class and a corresponding setup in the ’MlInputData’ class) into a specific folder on the hard drive, requiring no data normalization. The class methods operate on an absolute folder path stored in a private ’readonly’ variable of type string named ’FolderPath’. Names of the public methods implemented are:

Save

On method call a ’MlInputData’ object is provided, using the ’BinaryFormatter’ the object is saved to the folder specified by the ’FolderPath’ variable. The filename is created by using the current system time and formatting it in a Unix time style format, concatenated with the EHR and literature ID, all separated by underscores and using ’.bin’ as file extension.

Delete

Invocation parameter is an ’MlInputData’ object. Iterating over all files in the directory path saved in the ’FolderPath’ variable, comparing each filename to the EHR and literature ID of the ’MlInputData’ matching objects are identified and deleted from disk.

Load

The method has no parameter and returns a list of ’MlInputData’ objects. Iterating over all files in the directory specified in the ’FolderPath’ variable and facilitating the ’BinaryFormatter’, ’MlInputData’ objects are loaded from diThe method has an integer parameter, the EHR ID, and returns one ’MlInputData’ object. Internally the method employs a ’for each’-loop to iterate over all file names in the folder where the ’MlInputData’ objects are stored on the hard disk. In the loops body, using a regular expression, it is decided wether or not the current file name matches the given EHR ID. As soon as a match is found, the file is loaded from disk. By asserting the UserVote attribute of the previously loaded ’MlInputData’ object, it is checked if an up-vote is found. In case the ’MlInputData’ object represents an up-vote, the object is returned to the caller. If no up-vote (or no ’MlInputData’ object) is found for the given EHR ID, the method returns ’null’.

Find Positive Vote by EHR ID

The method has an integer parameter, the EHR ID, and returns one ’MlInputData’ object. Internally the method employs a ’for each’-loop to iterate over all file names in the folder where the ’MlInputData’ objects are stored on the hard disk. In the loops body, using a regular expression, it is decided wether or not the current file name matches the given EHR ID. As soon as a match is found, the file is loaded from disk. By asserting the UserVote attribute of the previously loaded ’MlInputData’ object, it is checked if an up-vote is found. In case the ’MlInputData’ object represents an up-vote, the object is returned to the caller. If no up-vote (or no ’MlInputData’ object) is found for the given EHR ID, the method returns ’null’.

Preparing Data for Machine Learning

With regard to the possibility that in later program evolution the machine learning part may be implemented by an external program or another library, the transformation from ’MlInputData’ to matrixes takes place in three steps. Data for machine learning is prepared by static methods of a (static) utility class.

Creating Vectors

Two public methods are used, each operating on a list of ’MlInputData’ objects and returning a dictionary object where a ’String’ is used as key to a value of type integer array. The string value is the column headline while each individual value in the integer array indicates the presence (1) or absence (-1) of the headline for an ’MlInputData’ object. Both integer values occurring in the array originate from static ’readonly’ class variable to ease reconfiguring. Both methods described below use private methods, to enable re-usage of implemented functionality.

getEhrForML

First all MlInputDataObject's are filtered to include up votes only and clustered using the content of its EhrData attribute of type 'SearchField' by its 'FieldName' (an enumeration). For each 'FieldName' all values are collected and duplicate values are removed. Iterating over the duplicate free list of values, each MlInputDataObject representing a user vote is checked wether or not its 'EhrData' attribute (a list of 'SearchField') contains the current value. The outcome is saved in an integer array which is placed alongside the value used for checking in the target dictionary object, serving as the method's return value. Using a boolean parameter the method can be configured to include or exclude the EHR ID number.

getTermsForML

The functionality offered by the 'getTermsForML' method is almost identical to the above mentioned 'getEhrForML' method. Instead of operating on the 'SearchField' list (stored in the 'EhrData' attribute) as the 'getEhrForML' method does, the method operates on the 'LiteratureData' attribute, being a list of 'Term' objects. First all term labels of the up voted literature are collected in a list and duplicates are removed. Iterating the duplicate free list each 'MlInputDataObject' is checked wether or not its 'LiteratureData' attribute contains the current term. The outcome of the check is placed in an integer array which is saved alongside the value used for checking in the methods return value dictionary object.

By using both previously described methods, the list of 'MlInputData' objects is transformed into two dictionaries, each using a string as key originating from the individual 'Term' or 'SearchField' objects (of the 'MlInputData' objects) with its corresponding occurrence as value being an integer array. The dictionary containing data from the EHR is ultimately used as input data for machine learning, while the one containing data from the literature is the expected output for machine learning.

Creating Matrixes

Comparable to vector transposition, an additional transformation of each individual dictionary takes place by facilitating a static utility method. On call the method receives the dictionary containing string headlines as keys and a corresponding integer array as value. Return value is a 2D integer 'list' (a list of lists containing integer values). Facilitating two nested loops, first the data of the integer arrays is iterated (the dictionary values) and second the string keys (column headlines) of the dictionary. In the inner loops body the individual data column is accessed and saved into the target 2D integer 'list'.

Value Exclusion, Casting and Normalization

As different machine learning implementations require different data types (and value ranges), as well as the fact that on prediction only the EHR data for prediction needs to be given to the mechanism, an additional transformation step takes place, based on the 2D integer 'list'.
Employing two nested loops on the 2D integer 'list', first the EHR/Literature ID is excluded. On element level in the inner loops body, the remaining values are checked for value ranges; in case a match occurs the value is replaced or casted to the target datatype. The result is a 2D array of the target datatype which is returned to the invoker.

Machine Learning Prediction

An out of the box C# implementation of an multi-label (ML) support vector machine (SVM) includeing a learn method is offered by the 'Accord.Net' library (details about a SVM can be found in 3.1, Machine Learning). On prediction method call a string list named EhrTermList containing all value attributes of the 'SearchField' Class' is provided alongside the EHR ID. The 'SearchField' objects originate from the EHR by employing the Semantic EHR Field Mapping (described in subsection 6.2.1). The prediction method loads the trained ML SVM from hard disk and stores the machine in a variable of type 'MultilabelSupportVectorMachine' named 'Machine' and checks, wether or not at least one string of EhrTermList (provided on invocation) is known by the ML SVM, using string comparison with ignore case option on the (EHR) input string list retrieved alongside the ML SVM.
In case the EhrTermList contains none of the input headlines the ML SVM is trained with, the method returns null.
In case at least one string of the EhrTermList is known by the ML SVM, a utility function is invoked, producing a double array named 'Input' where each cell indicates the presence (1) or absence (-1) of the known ML SVM input string in the provided string list. All strings unknown by the ML SVM are omitted. This double array is used as input for prediction. Prior to the actual prediction call (shown in code listing 6.15) an auxiliary array is created to store the ML SVM's "raw" (not 1,-1 normalized) prediction for later use. The result of the ML SVM prediction is saved in an integer array.

```
...
int[] MlComputedValueArray = Machine.Compute(Input, out MlResponse);
...
```

Listing 6.15: Multilabel Support Vector Machine Prediction

After the prediction call to the ML SVM, the provided EHR ID is used during a call to the user feedback store's method named 'Find Positive Vote by EHR ID' (the method is described in 6.2.3). If the method invocation returns a 'MlInputData' object different from null, a utility method is employed to transform the object's EHR terms to an integer array using the known ML SVM input string list. The resulting array is used as expected output during creation of the Confusion Matrix object, while the prediction result is used as predicted input. In case the feedback store does not contain positive feedback to the EHR, no Confusion Matrix object is created, the corresponding variables (Confusion Matrix and expected output) is set to null.
The prediction method ends by returning both prediction arrays (normalized and not normalized), the Confusion Matrix as well as the term labels (retrieved alongside the ML SVM) and the expected output in a container object named 'MlOutput'.
The 'MlOutput' object is inspected and two string lists are created, containing the query terms as well as the terms to highlight. Both are handed over to the query creation (described subsection 6.2.4, Query Creation). Starting with a ML SVN trained on all available test data and invoking it for a known EHR, search term predictions are given. The terms are 'AND' connected to form the query.
If the resulting query is to specific – no literature IDs are found while querying - the not normalized prediction array is taken into account. A tuple list is created containing the not normalized positive numeric prediction value and the corresponding search term. This list is sorted according to the tuple's prediction value in descending order. Employing a while-loop, the term with the lowest prediction value is removed and another query for literature IDs is performed. The while-loop ends if literature IDs are found or the tuple list does not contain any (more) values.
If literature IDs where found, the corresponding literature is retrieved. In case no literature IDs are found, an empty list of literature objects is returned to the invoker.
Details of a test run can be found in appendix E, Test Run: Semantic Query Term Identification.

6.2.4 Query Creation

The query creation as described in subsection 5.2.5, Query Creation, uses the output from one of the semantic query term identification modules to create a query. While iterating the results of the query term identification, each query term is interconnected using ' AND ', forming the query string. Aside the query string creation, the search terms to highlight in the fitting literature (being the second result of the semantic query term identification) are concatenated via ','. Both strings are returned to the invoker.

6.2.5 Quering PubMed

To query PubMed NLM's 'E-utilities' endpoints are used as described in subsection 3.4.1, PubMed API. All communication with the API is handled with private methods of the 'PubMedApiClient' class. An overview of the communication between class and the APIs shows figure 6.2. After instantiation, the class offers three public methods, each relying on private methods.

Get Literature

The 'getLiterature' method expects a query string on invocation. An optional integer parameter is offered, specifying the amount of literature to be found, its default value is 10.

Result of the method call is an 'IList' of 'Literature' objects. Details on the literature object can be found in subsection 5.1.1, Literature Class. If no literature is found for the query, an empty list is returned to the caller.

Get Literature IDs

All parameters of the 'getLiteratureIds' method are identical to the above mentioned 'getLiterature' method.

The result is an 'IList' of 'string' objects containing literature IDs. In case no literature is found for the given query, an empty list is returned.

Get Literature For ID

The cunctionality the method offers is obtaining literature for a literature ID in string format.

Resulting in a 'Literature' object. If no literature for the ID is available null is returned.

Communication with the 'E-utilities' endpoints is first described and than shown in 6.2. The query string is given via the HTTP GET method to the 'esearch' API returning a JSON formatted list of literature IDs. Details for each literature ID such as title, abstract and author are retrieved in XML format using the 'efetch' API per literature ID. Both calls to the 'E-utilities' as well as the JSON and XML parsing are explained after the diagram.

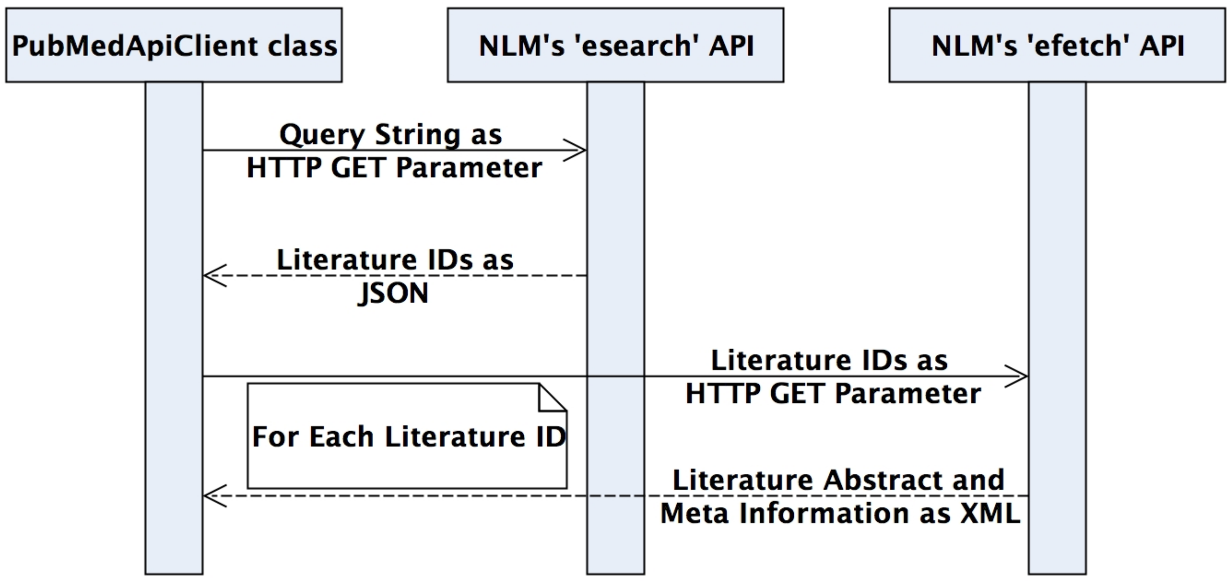

PubMedApiClient class
NLM's 'esearch' API
NLM's 'efetch' API
Query String as HTTP GET Parameter
Literature IDs as JSON
Literature IDs as HTTP GET Parameter
For Each Literature ID
Literature Abstract and Meta Information as XML

Figure 6.2: UML Sequence Diagram - PubMed Querying

Literature ID Retrieval

On call of the 'searchForLiteratureIds' method shown in listing 6.16 the query string and the desired maximum literature result count are handed over (line 1). After creation of the 'ReturnValue' and the 'WebClient' in line 3 and 4, the 'WebClient' is configured. Line 4 shows the 'using' statement, it ensures the disposal of all objects even if an error occurs. In line 6 the encoding is set to UTF-8, because the query string can contain special characters. UTF-8 is supported by the 'esearch' endpoint. Starting from line 7 - 10 the parameters are set, beginning with the 'db' parameter set to 'pubmed', indicating that PubMed is queried. Second parameter is the 'retmode' parameter set to 'json' (xml is not supported by 'esearch'). Third parameter named 'term' is set to the 'Query' string. The very last parameter is the 'retmax' parameter, specifying the maximum count of literature IDs to be retrieved. In line 11 the 'DownloadString' method of the 'WebClient' is invoked, providing the 'SearchTermUrl' containing the URL of the 'esearch' API. The 'SearchTermUrl' is a readonly class variable of type string with private visibility. Result of the call is a list of literature IDs in JSON format, stored in the 'ReturnValue' and returned to the caller in line 13.

```
 1 private string searchForLiteratureIds(string Query, int MaxResultSize)
 2 {
 3   string ReturnValue = "";
 4   using( WebClient WebClient = new WebClient())
 5   {
 6     WebClient.Encoding = System.Text.Encoding.UTF8;
 7     WebClient.QueryString.Add("db", "pubmed");
 8     WebClient.QueryString.Add("retmode", "json");
 9     WebClient.QueryString.Add("term", Query);
10     WebClient.QueryString.Add("retmax", MaxResultSize + "");
```

```
    ReturnValue = WebClient.DownloadString(SearchTermUrl);
  }
  return ReturnValue;
}
```

Listing 6.16: Literature ID Retrieval - Method

Literature ID extraction from JSON

Employing the 'Json.NET' library, the content of the string variable with the literature IDs in JSON format is transformed into an 'IList' of strings using the 'extractLiteratureIDs' method shown in 6.17.
Line 3 shows the creation of the return value as 'IList' of type string with name 'DocumentIds'. Prior to JSON parsing the string provided at method invocation is checked wether or not it contains data and the expected JSON token named 'idlist' with the literature IDs (line 4). All following takes places only if the check in linve 4 is positive.
In line 6 a variable of type 'JObject' with name 'Json' is created. The variable is assigned with the object returned from the 'JObject.Parse' call, handing over the literature IDs provided at invocation in JSON format stored in the 'LiteratureIdsJson' variable of type string. Variables of type 'JObject' allow a fluent invocation of the 'SelectToken' method as shown in line 7,8 where the different tokens (esearchresult, idlist) along the path to the ID list are traversed. In line 9 by using the 'Select' method in combination with a lambda expression to cast the variable 's' to a string, an implementation of the 'IEnumerable' interface is given, offering the 'ToList' method. The result is the list of literature IDs.
Case the check was not positive the return values is assigned with a newly created list of type string in line 13.
In any case an 'IList' of type string is returned in line 15.

```
 1 private IList<string> extractLiteratureIDs(string LiteratureIdsJson)
 2 {
 3   IList<string> DocumentIds;
 4   if (!String.IsNullOrEmpty(LiteratureIdsJson) &&
         LiteratureIdsJson.Contains("idlist"))
 5   {
 6     JObject Json = JObject.Parse(LiteratureIdsJson);
 7     DocumentIds = Json.SelectToken("esearchresult").
 8             SelectToken("idlist").
 9             Select(s => (string)s).ToList();
10   }
11   else
12   {
13     DocumentIds = new List<string>();
14   }
15   return DocumentIds;
}
```

Listing 6.17: Literature ID extraction from JSON

Literature Details Retrieval

For each previously retrieved literature ID the 'getLiteratureDetails' method shown in 6.18 is invoked. The method is comparable to the 'searchForLiteratureIds' method in 6.16. However, the method parameters and the HTTP request parameters as well as the URL differ.
On method invocation the literature ID is provided as string variable named 'Id' (line 1). In line 3 the 'ReturnValue' variable of type string is created and initialized. The 'using' statement in line 4 ensures proper disposal of the 'WebClient' object even if an error occurs. A class instance of the 'WebClient' is created and stored in the 'WebClient' variable (line 4). In line 6 the encoding of the 'WebClient' is set to UTF-8, because literature details can contain special characters. From line 7 – 10 all required parameters for the 'efetch' API to query PubMed are configured. Setting the 'db' parameter to 'pubmed' (line 7) via the 'retmode' parameter in line 8, to 'xml' specifying the expected return format (only xml and plain text are supported) via the 'rettype' parameter indicating the information type of the request in line 9 to the final 'id' parameter in line 10 to the literature ID. The 'efetch' API call using HTTP GET with all prior configured parameters takes place in line 11 by invoking the 'DownloadString' method handing over the 'SearchAbstractUrl', containing the URL to the 'efetch' API. The 'SearchAbstractUrl' variable is a private readonly class variable, type string. Result of the API call is a string containing the literature details in XML format. It is saved in the 'ReturnValue' variable (line 10) and returned to the invoker in line 13.

```
 1 private string getLiteratureDetails(string Id)
 2 {
 3   string ReturnValue = "";
 4   using (WebClient WebClient = new WebClient())
 5   {
 6     WebClient.Encoding = System.Text.Encoding.UTF8;
 7     WebClient.QueryString.Add("db", "pubmed");
 8     WebClient.QueryString.Add("retmode", "xml");
 9     WebClient.QueryString.Add("rettype", "abstract");
10     WebClient.QueryString.Add("id", Id);
11     ReturnValue = WebClient.DownloadString(SearchAbstractUrl);
12   }
13   return ReturnValue;
14 }
```

Listing 6.18: Literature Details Retrieval - Method

Literature Details Extraction from XML

Literature details are mapped from the XML download format to the literature class using a parser relying on XPath. Details on the literature class can be found in subsection 5.1.1, Literature Class. Prior to explaining the mapping shown in listing 6.19, the parser named 'PubMedXmlParser' is explained.
On instantiation, the parsers constructor expects the string containing the XML to be parsed. Each value for the six literature class attributes can be retrieved from the parser by using public methods per attribute. Internally the parser invokes for 5 out of 6 attributes one private method with a distinct XPath query to directly retrieve the desired value.
Only the authors attribute of the literature class is extracted differently because author information, being a complex attribute requires more complex XPath query and target datatype creation than any other attribute. XPath is still used in the author information extraction, however only as entry point discovery mechanism. As the author information entry point is found by the XPath query, standard control flow mechanisms (loops and conditions) combined with string comparisons are used to get the values.
The parser internal private utility method's source code can be found in appendix D.4, PubMed XML Parser Based on XPath. It is capable of parsing XPath single element selection queries as well as queries containing the union operator ('|'), concatenating the content of both the operator sides with specified separators while keeping the element order. Table 6.2, Literature Details XPath Queries shows the queries handled by the method with example result data.

XPath Query	Example Parser Output
//AbstractText	Cytotoxic T-lymphocyte-associated antigen 4 (CTLA-4) ...
//PublicationType	Clinical Trial, Phase II
//PublicationStatus	ppublish
//ArticleTitle	Ipilimumab plus sargramostim vs ipilimumab ...
//DateCreated/Year | //DateCreated/Month | //DateCreated/Day	2014 11 05

Table 6.2: Literature Details XPath Queries

All XPath queries are stored centrally in a public static readonly dictionary using an enumeration as key and string as value inside the static class named 'PubMedXmlParserConfig'. Allowing named access from the individual get attribute method to the queries as well as easy maintenance.
The source code shown in listing 6.19 uses the parser to map literature information retrieved in XML to the literature class. On invocation the literature ID is provided in string format (line 1). Using the previously explained 'getLiteratureDetails' method (6.18) the literature's XML string is retrieved in line 3. In line 5 an instance of the 'PubMedXmlParser' class is created and saved in the local variable named 'Parser'. Calling the parser's constructor the 'DetailsXmlString' is provided. Starting from line 7 to 12 all values for the literature

class attributes are retrieved from the parser and saved in a corresponding local variable with appropriate datatype. In line 14 a new instance of the literature class is created. On its constructor invocation all local variables containing the parsed values are provided. The resulting literature object is returned to the invoker in line 14.

```
1  private Literature queryLiteratureById(string Id)
2  {
3    string DetailsXmlString = getLiteratureDetails(Id);
4
5    PubMedXmlParser Parser = new PubMedXmlParser(DetailsXmlString);
6
7    IList<LitratureResultAuthorInformation> AuthorInformationList =
         Parser.getAuthors();
8    string AbstractText = Parser.getAbstractText();
9    string PublicationType = Parser.getPublicationType();
10   string PublicationStatus = Parser.getPublicationStatus();
11   string ArticleTitle = Parser.getArticleTitle();
12   string DateCreated = Parser.getDateCreated();
13
14   return new Literature(Id, AuthorInformationList, AbstractText,
         PublicationType, PublicationStatus, ArticleTitle, DateCreated);
15 }
```

Listing 6.19: Query Literature By ID - Method

Local Literature Cache

Because the literature store is an external dependency, to abide the recommendations of NLM regarding API usage as best as possible and to decrease response time for multiple users requesting literature for the same EHR, a local in memory literature object cache is implemented. Instead of calling the private queryLiteratureById method directly inside the public method getLiteratureForId, a lookup in the local cache takes place. Once the literature ID is in the cache the object is retrieved from the cache, no invocation of the queryLiteratureById method takes place. As long as the cache's maximum size is not reached, literature objects are stored in the cache. The cache is implemented by a private static C# 'Dictionary' using a string as key (for the literature ID) and the literature object as value. Its maximum size is set to 1000 and all of its content is in memory only and not saved or filled from disk during application startup or shutdown.

6.2.6 Highlights of the Literature View Component

The literature view component comprises of different Javascript functions. A screenshot of one literature view component, created by the Javascript functions is shown in 5.2.8. As soon as the EHR is opened in review mode, an asynchronous HTTP post call to the back-end takes place, handing over the EHR ID and the expected amount of literature. The convenient post function is offered by the jQuery library returning a promise. As long as the back-end processes the request, a loading icon is displayed in the webpage's literature area. When the response of the back-end arrives (facilitating the jQuery 'when' construct on the HTTP post call object) the result in JSON format is iterated and HTML elements are created for each literature object, each element is appended to a local variable. All HTML elements use a fitting CSS (Cascading Style Sheet) class of the 'Bootstrap' library to ensure a responsive design. Prior to displaying the created HTML elements (stored in the local variable) to the user, the loading icon is hidden and all content of the local variable is appended to the webpage's literature area.

Author information

Each literature objects author information is processed by a function, which creates a convenient author information string. Each authors initial followed by a blank is appended to his surename and all such tuples are concatenated using the a comma. All 'AuthorInformation' per literature object are shown to the users at the push of a button.

Search Term Highlighting

The back-end response contains the employed search terms in an array. Identical to the term part highlighting using a regular expression as stated in subsection 6.1.6, Highlights of the Term View Component each literature's title and abstract are inspected for search terms and highlighting takes place.

Literature Votes

While iterating the literature objects an additional HTML element per literature is created to enable users to up or down vote literature objects. The voting response is shown to the user in hiding the corresponding HTML vote element and displaying the fitting "You voted" HTML element for 5 seconds by using the javascript 'setTimeout' function. Before displaying the vote feedback, per vote a HTTP post call to the back-end takes place, handing over the literature ID and the EHR ID as well as a boolean value indicating a positive or negative vote.

NLM Disclaimer

As soon as all literature objects are converted to HTML elements, a static link to the NLM disclaimer is appended to the variable which is to be appended to the literature area.

Response Time

On invocation of the HTTP post to the back-end asking for literature, the system time is stored in a local variable. As the very last step of the literature object transformation from JSON to HTML elements, the system time is retrieved again. Calculating the difference between those two time steps, the response time is calculated and appended to the variable which is appended to the literature area.

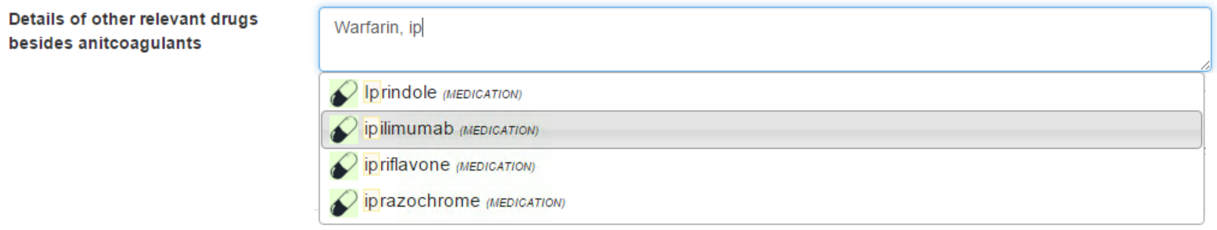
Chapter 7

Evaluation

Bases for the evaluation is the current application with its two new services. Most evaluations rely on screenshots of the functionality in question. All collected values and screenshots are dumps of the application running on a private server inside a virtual machine relying on Sun Virtual Box 5.0 on top of Debian GNU Linux 8.0. Inside the virtual machine Microsoft Windows 8.1, Microsoft Visual Studio 2013, Sun Java 1.8 and Apache Solr 5.1 is installed and configured. During data collection other virtual machines are running. If the evaluations are performed on a more capable host or e.g. without virtualization the results considering the response time ought to be better.

7.1 Functional Requirements

1. Medical context fitting terminology with two prototpyes
   The screenshot below (7.1) shows the EHR's 'Details of other relevant drugs besides Anticoagulants' field with the terminology service switched on. Terminology fitting to the field is offered on user input. Only terms of the medication category are suggested, matching the first half of terminology service's functional requirement.



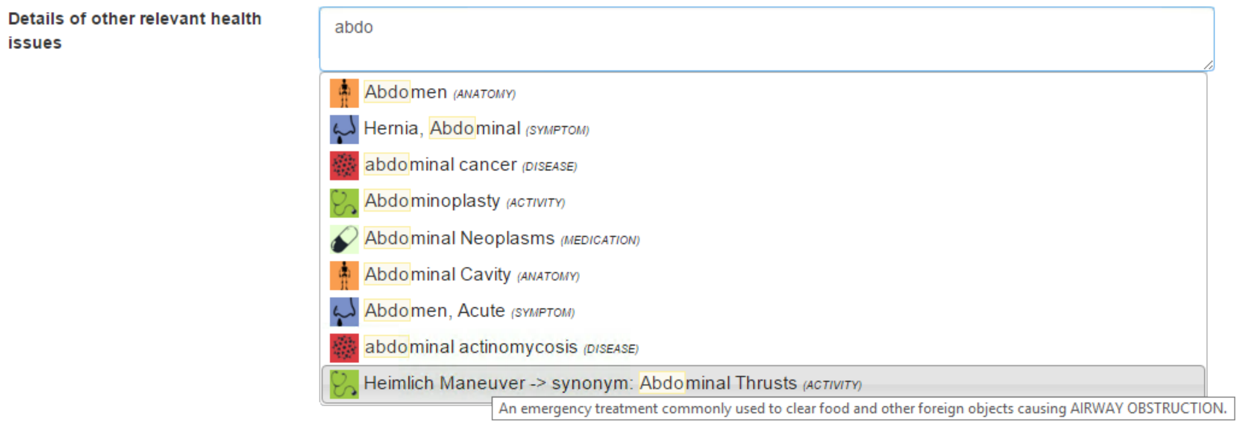
Details of other relevant drugs besides anitcoagulants

Warfarin, ip

Iprindole (MEDICATION)
ipilimumab (MEDICATION)
ipriflavone (MEDICATION)
iprazochrome (MEDICATION)

Figure 7.1: Screenshot Terminology Service - EHR Field: Details of Other Relevant Drugs Besides Anticoagulants

As shown in screenshot 7.2 the terminology service is enabled for the EHR's 'Details of other relevant health issues' field. Terminology of all categories are offered on user input, meeting the second half of the terminology service's functional requirement.



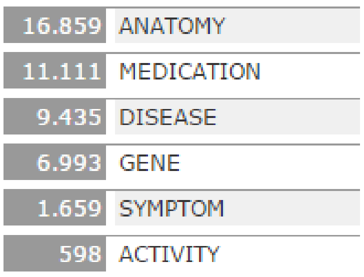
Details of other relevant health issues

abdo

Abdomen (ANATOMY)
Hernia, Abdominal (SYMPTOM)
abdominal cancer (DISEASE)
Abdominoplasty (ACTIVITY)
Abdominal Neoplasms (MEDICATION)
Abdominal Cavity (ANATOMY)
Abdomen, Acute (SYMPTOM)
abdominal actinomycosis (DISEASE)
Heimlich Maneuver -> synonym: Abdominal Thrusts (ACTIVITY)
An emergency treatment commonly used to clear food and other foreign objects causing AIRWAY OBSTRUCTION.

Figure 7.2: Screenshot Terminology Service - EHR Field: Details of Other Relevant Health Issues

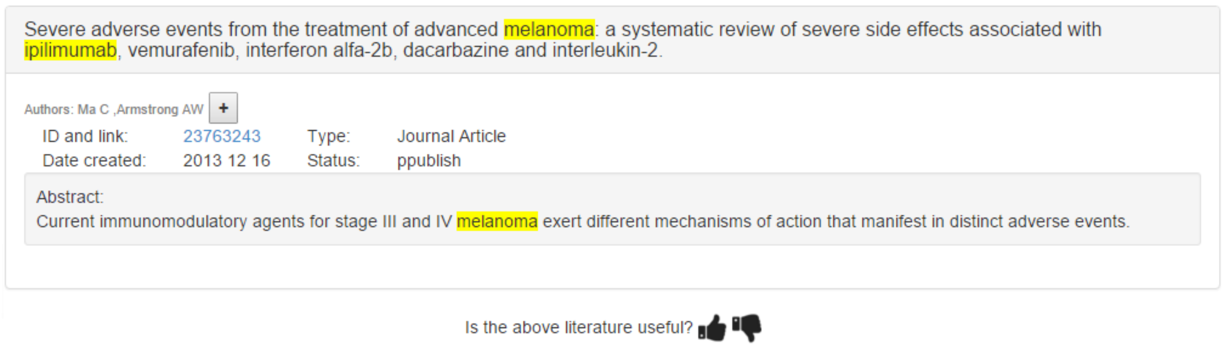
46655 terms are stored in the term store, in the six medical categories originating from ten medical ontologies. The term count per category can be seen in figure 7.3, a screenshot of the Solr Admin Web interface. As indicated by the screenshots, the necessary requirement is met.



16.859	ANATOMY
11.111	MEDICATION
9.435	DISEASE
6.993	GENE
1.659	SYMPTOM
598	ACTIVITY

Figure 7.3: Screenshot Terms per Category - Solr Admin Web Interface

2. Useful and accurately fitting literature based on the EHR must be displayed.
   The literature's suitableness can be assessed only by a domain expert with knowledge of the EHR in question. However, based on the test data provided by NSilico the following check is possible:
   Applying the conventional (manual) way, a domain expert from NSilico traced down five medical literatures for a known EHR test record, where the literature service found a total of twenty four medical literatures, four teen of them from the rule approach and ten from the machine learning approach (details about the queries performed can be found in appendix E, Test Run: Semantic Query Term Identification). One of the manualy found literatures is at 10th position of the literature service's (automatic) result. The screenshot shown in 7.4 displays the literature in question.



Severe adverse events from the treatment of advanced melanoma: a systematic review of severe side effects associated with ipilimumab, vemurafenib, interferon alfa-2b, dacarbazine and interleukin-2.

Authors: Ma C ,Armstrong AW +

ID and link:	23763243	Type:	Journal Article
Date created:	2013 12 16	Status:	ppublish

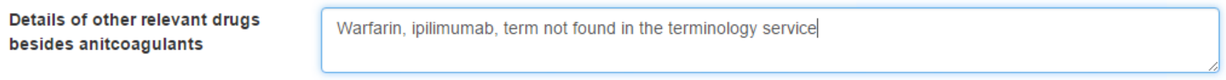
Abstract:
Current immunomodulatory agents for stage III and IV melanoma exert different mechanisms of action that manifest in distinct adverse events.

Is the above literature useful?

Figure 7.4: Screenshot Literature Service Result - Useful and Accurately EHR Fitting Literature

   Hence, the requirement to show useful EHR fitting literature is met.

3. Usability of new modules
   The terminology service does not interfere with the user experience. A user starts typing and once the EHR field is configured to use the terminology service, suggestions are provided. As the user is looking for a term not provided by the terminology service, it is still possible to enter the word, as shown by the user input 'term not found in the terminology service' in screenshot 7.5.


Details of other relevant drugs besides anitcoagulants

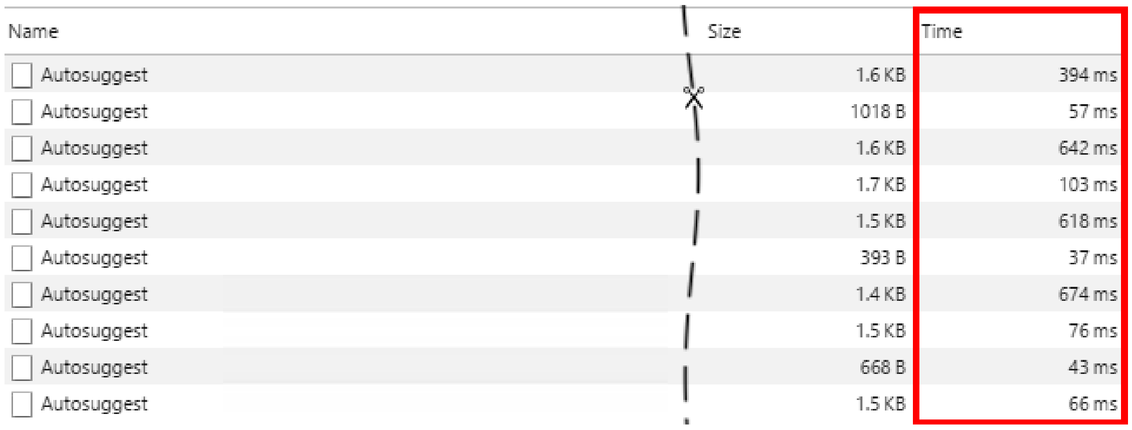
Warfarin, ipilimumab, term not found in the terminology service

Figure 7.5: Screenshot Terminology Service - Input of Unknown Terms is Supported

The literature request is triggered asynchronously on page load as the EHR is requested in 'view' mode. While the EHR is looked at, relevant literature is searched for and a loading icon is displayed. As soon as the literature results are available, a part of the view is updated, hiding the loading icon and showing the literatures discovered. No additional user action is required. However, for the machine learning part of the literature service, a user can vote up or down literature as shown for one literature object near the bottom of screenshot 7.4.

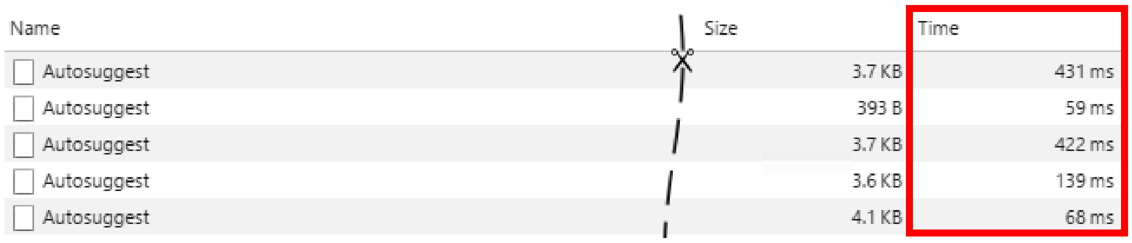
7.2 Non Functional Requirements

1. The terminology must be offered instantly (below 1 s)
   Enabling the web browser's debug window during the terminology service usage in 'Medical context fitting terminology with two prototpyes' the service response times is shown. Screenshots 7.6 and 7.7 show corresponding values. For the 'Details of other relevant drugs besides Anticoagulants' field the terminology service's maximum response time is 674 ms. Responses for the 'Details of other relevant health issues' field arrive in maximum 422 ms, both values are below the 1 s. non functional (timing) requirement.



Name	Size	Time
Autosuggest	1.6 KB	394 ms
Autosuggest	1018 B	57 ms
Autosuggest	1.6 KB	642 ms
Autosuggest	1.7 KB	103 ms
Autosuggest	1.5 KB	618 ms
Autosuggest	393 B	37 ms
Autosuggest	1.4 KB	674 ms
Autosuggest	1.5 KB	76 ms
Autosuggest	668 B	43 ms
Autosuggest	1.5 KB	66 ms

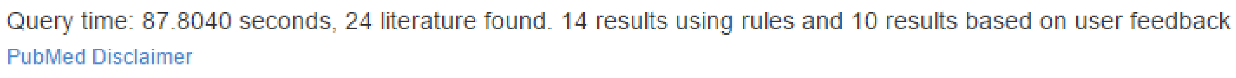
Figure 7.6: Screenshot Terminology Service - EHR Field: Details of Other Relevant Drugs Besides Anticoagulants - Response Time



Name	Size	Time
Autosuggest	3.7 KB	431 ms
Autosuggest	393 B	59 ms
Autosuggest	3.7 KB	422 ms
Autosuggest	3.6 KB	139 ms
Autosuggest	4.1 KB	68 ms

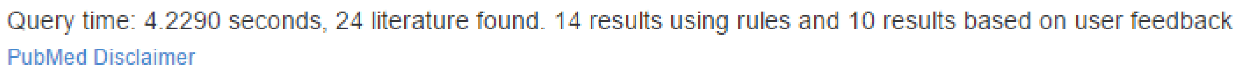
Figure 7.7: Screenshot Terminology Service - EHR Field: Details of Other Relevant Health Issues - Response Time

2. Collecting ten context relevant literatures and displaying them must in no case interfere with the normal EHR processing time during physicians review (below 5 min).
   The time required by the asynchronous literature search to find literature depends on multiple factors. The factor with the highest time impact is the response time of the external literature store. The view part of the literature service shows the overall response time for one EHR. Shown in screenshot 7.8 the first literature request for an EHR after application start amounts to ~88 s. To compensate this behavior all displayed literature is cached, as long as the literature cache size is not exceeded in the application. If an EHR is opened a second time, the most part of the literature request is answered by the literature cache, resulting in a response time of ~4 s as shown in screenshot 7.9. In the worst case the literature cache is full or empty, resulting in the highest response time, however both values are less than the required maximum of 5 minutes.



Query time: 87.8040 seconds, 24 literature found. 14 results using rules and 10 results based on user feedback
PubMed Disclaimer

Figure 7.8: Screenshot Literature Service - Intitial Loading - Response Time



Query time: 4.2290 seconds, 24 literature found. 14 results using rules and 10 results based on user feedback
PubMed Disclaimer

Figure 7.9: Screenshot Literature Service - Reloading - Response Time

3. All new modules must be easy to maintain and extend.
   To prove that both the terminology service and the literature service are easy to maintain and extend, three most likely cases per service are evaluated as follows:

Terminology Service

Case 1: Add new Ontology

A new ontology can be added by implementing an appropriate loader class and add its instantiation and load method call to the existing terminology ETL loader method. Two loader implementations are described in subsection 6.1.3, ETL. As soon as the ETL is triggered again, the new ontology is included. Additional requirements are: A IList of 'Term' objects is offered by calling a fitting method of the new loader. Each Term object created by the new loader must fill the required attributes of the Term class as described in subsection 4.2.3, Term Data Model being 'label', 'category', 'TermId' and 'source'. Should the ontology not be part of an already known source, the new source must be added to the source enumeration and to the default score method.

Case 2: Add new Category

A new category can be subjoined by adding its name to the categories enumeration, adjusting the default score method and providing an appropriate PNG icon to the view component. In addition, the Javascript view function must be adjusted to incorporate the new category weight both on function call and in the request to the back-end. The controller and the terminology service have to be extended to include the new category weight as parameter, both are located in the back-end. In the terminology service method the new category must be added to the category configuration mechanism. By setting a default value for all new parameters, the impact of method signature changes can be minimized.

Case 3: Add new EHR field

A new EHR field can be added by calling the terminology service's Javascript view function, providing the HTML input elements ID alongside the desired maximum suggestion count as well as the category weights. Per EHR field one function call is required, its signature being shown in subsection 6.1.6, Highlights of the Term View Component.

Literature Service

Case 1: Add new EHR field

In case a new medication field is added to the EHR, only the semantic EHR field mapper needs to be extended to include a mapping for the new field. Two example mappings are shown in subsection 6.2.1, Semantic EHR Field Mapping.

Case 2: Add new EHR field category

Should the new EHR field disposes of a new EHR category - aside the semantic EHR field mapper - the rule based semantic query creation needs to be adjusted accordingly. Either the new category is added to the exclude list or a new rule needs to be created. Two example rules are shown in subsection 6.2.2, Rule Based Semantic Query Term Identification.

Case 3: XML format of the Literature changes

Case the result format from NLM's PubMed API (the literature store) changes, the XPath expressions used for the mapping between XML result of the store and the literature class needs to be changed. Depending on the change magnitude, the XPath parser needs to be adjusted too. However the changes are local to the 'PubMedXmlParser' class or even its configuration class only, both described in subsection 6.2.5, Literature Details Extraction from XML

Chapter 8

Related Work

8.1 Medical Term and Ontology Categories

The term and ontology categories of this thesis are closely related to the ’content’ categories found in the "Encyclopedia of Database Systems" ([25] p. 362). However, the amount of different ontologies evaluated are fewer than in the thesis, but the NLM’s UMLS, (Unified Medical Language System) being itself a collection of ontologies, is part of the article’s comparison. In addition the article does not contain any actual test regarding the ontologies’ content.

8.2 Diversity and Relevance of the Term Suggestion

There exists a comparable system for a german library[54]. Diversity and relevance of literature search results are evaluated using a sophisticated algorithm and then displayed applying a special user interface component called ’topic pie’. However, it is to be found in another application domain and covers a different use case (direct user interaction to search for literature) as well as not using context aware terminology suggestion to the extent as does the terminology service described in the thesis.

8.3 EHR Mapping, NLM’s MeSH and PubMed API

Feng et. al. published a paper concerning ’the quantitative measurement of clinic-genomic association for colorectal cancer using literature mining and google-distance algorithm’ [55]. Similar to the thesis a normed vocabulary is used (NLM’s UMLS), where MeSH is included. Instead of using EHR field categories as the thesis does, the EHR is mapped using the ’MetaMap’ program enabling text concept recognition of known concepts. Another similarity to the thesis is the use of NLM’s PubMed API but the article describes the usage of the ’esearch’ API only, where the thesis uses the ’efetch’ API as well because the publication focuses more on association mining and does not include an automatic search for literature as the thesis does.

8.4 Literature MeSH Category Prediction

Pitigala et. al. wrote an article concerning "A comparative study of text classification approaches for personalized retrieval in PubMed" [56]. The idea of the articles "[...]text classification systems is to automatically assign individual electronic documents to one or more categories based on their contents" ([56] p. 919). Like the literature services machine learning data collection feature (described in 6.2.3, Machine Learning Based Semantic Query Term Identification) the article mentions the use of the literature's webpage, however they retrieve the whole article from the journals webpage where the functionality implemented in the theses extracts keywords and other terms per literature. In addition the article depict different transformation steps, including stemming to extract terms from the article, where the thesis uses the terminology service to identify literature search terms. The article describes the evaluation of a SVM and Naïve Bayes to predict MeSH (sub) categories for literature. One of the articles conclusion is that "In all the experiments, the Naïve Bayes method outperforms the SVM method" ([56] p. 921). However the prediction evaluation is difficult to compare to the implementation in the thesis, because the thesis predictions are special to an EHR and not to "broad" MeSH categories.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

Both the terminology service and the literature service implemented in this thesis match the requirements described in chapter 2, Problem Statement.

From the data point of view, the core of the terminology service is formed by the terminology store filled on ETL time with ten ontologies fitting to the six medical term categories as mentioned in chapter 4, Terminology Service and subsection 6.1, Terminology Service, producing a medical vocabulary of 46655 terms. All used ontologies are selected by checking the existence of known EHR terms, originating from demo EHRs (the table showing all evaluated terms and ontologies can be found in appendix A, Medical Ontologies). Employing semantic mechanisms like the duplicate term removal (described in 4.2.4, Duplicate Term Removal) and the result optimization (explained in 4.3.1, Semantic Result Optimization) as well as a convenient user interface component (shown in 4.3.2, User Interface) and the services invocation function for the front-end (the methods signature is shown in 6.1.6, Highlights of the Term View Component), users are supplied with EHR field fitting medical terms. As proved in the evaluation chapter 7, Evaluation, the terminology service matches both its functional and non functional requirements. Extensions to the terminology service are described in subsection 9.2, Extending the Terminology Service.

The task solved via the literature service is to display EHR context specific literature to users. Checking, wether or not the requirement is met, is a challenge in itself and can be evaluated by a domain expert only as stated during the medical search engine selection (subsection 5.2.7, Medical Literature Search Engine Selection) and the evaluation (chapter 7, Evaluation). The need for literature is EHR specific and, in addition, different physicians have most likely distinct ideas what literature is useful. The first part of the previous argument is tackled by the literature services static rules, based on provided demo data, used for query term identification, while the second part of the argument is mastered by employing machine learning based on user feedback. A possible extension of the literature service (described in subsection 9.3, Extending the Literature Service) shows how user personal literature suggestions could be implemented. The output of the semantic query term identification (based on rules and machine learning) is employed to form a query to NLM's PubMed API used as literature store. As shown in the evaluation (chapter 7, Evaluation), helpful literature is provided. At the time being, the machine learning part is able to predict search terms. However, to futher assert the capability of the employed multi-label Support Vector Machine for query term predictions, more user feedback is required. The results of one machine learning prediction is shown in appendix E, Test Run: Semantic Query Term Identification

The previous paragraphs support the author's opinion that a comprehensive test data collection is invaluable, because the terms are the most important criteria of the ontology selection for the terminology service and basis for the literature services semantic query term identification - for both the rule appliance and the machine learning based approach (the test data is used as initial training set).

From my point of view the notable results of both the literature service and the terminology service are only possible because of the thoroughly performed evaluations and tests of the ontologies and the medical search engine comparisons as well as the careful design and implementation of all components. To my mind, the machine learning approach of the semantic query term identification will produce solid results, even better ones as soon as it is provided with more data. This assumption is circumstanciated by the test run's result, described in appendix E, Test Run: Semantic Query Term Identification.

I experienced the work at this thesis as an interesting task. All modules created have the pontential to help improve medical diagnosis and therapy.
It's great, because it works!

9.2 Extending the Terminology Service

In this section extensions to the terminology service are described. Prior to further plan or implement any of the stated extensions, chapter 4, Terminology Service, and the corresponding description of the services implementation in section 6.1, Terminology Service, should be consulted. While extending the service there is potential to re-use existing components.

9.2.1 Improve the Exclude Lists

At the moment the ETL uses an exclude list per ontology as well as a global exclude list. Based on domain expert's feedback the lists may be filled with more terms.

9.2.2 ETL Speedup

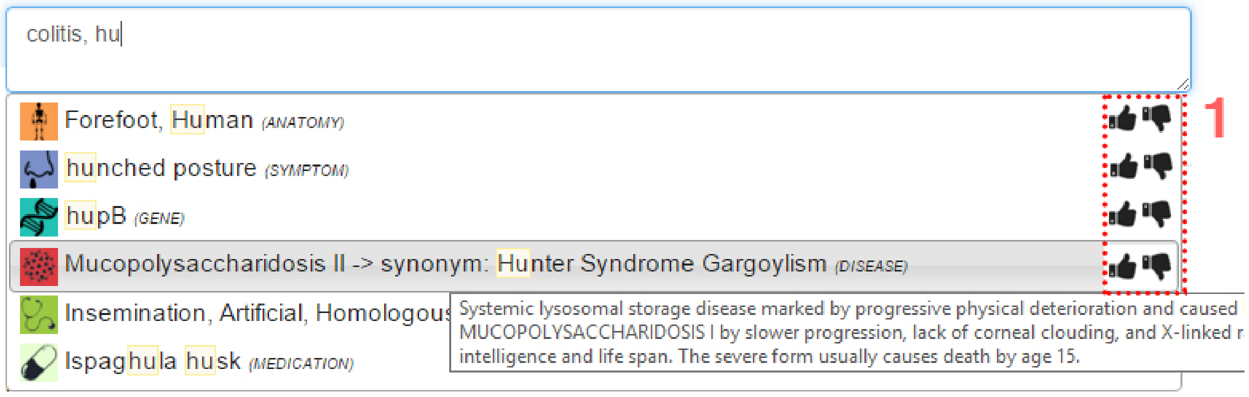
The ETL process is currently split into two parts, the time consuming extract and transform part and the fast load part, in between both parts all terms are saved into a file for decoupling (subsection 6.1.3, ETL). All Terms from the ontologies are sequentially extracted it could be done in parallel.

9.2.3 Add more Ontologies

As stated in 6.1.3, ETL the loader for the National Cancer Institute Thesaurus is not yet implemented. In addition, more MeSH subtrees could be added as well as other fitting ontologies. In the Evaluation Chapter - Non Functional Requirements - Terminology Service - Case 1: Add new Ontology, the process to add a new ontology is described in detail.

9.2.4 User Term Feedback

The terminology service provides suggestions based on the configured category weights, total suggestions count per EHR field and user input as described in e.g. 4, Terminology Service. Taking user feedback per term into account, the suggestions could be personalized. In addition, shortcomings from the static term exclude lists during ETL time (described in '6.1.3 ETL', subsection 'Global Term Label Exclude List') could be avoided by a long shot. To accomplish customized terminology suggestions both the front-end part as well as the back-end part of the terminology service needs to be extended.
In the front-end terminology display, an additional field would be required, e.g., on the right hand side of each individual term, as shown in figure 9.1.


colitis, hu
Forefoot, Human (ANATOMY)
hunched posture (SYMPTOM)
hupB (GENE)
Mucopolysaccharidosis II -> synonym: Hunter Syndrome Gargoylism (DISEASE)
Insemination, Artificial, Homologous
Ispaghula husk (MEDICATION)
1

Systemic lysosomal storage disease marked by progressive physical deterioration and caused
MUCOPOLYSACCHARIDOSIS I by slower progression, lack of corneal clouding, and X-linked r
intelligence and life span. The severe form usually causes death by age 15.

Figure 9.1: Proposal: Terminology Voting

By implementing the feature users are enabled (similar to the literature service) to up or down vote certain terms (figure 9.1, mark 1) influencing the individual term position or even exclude the term from future suggestions. One challenge regarding the user interface is separating term clicks into the term selection part and the term voting part. In addition, a

mechanism to delete votes would be required, hence, a user could vote a term up or down by accident. This may result in a personal terminology administration option which should be placed in the future user's admin panel to enable vote deletion.
On the back-end side, a user term feedback store is needed that persists the user's term feedback. Taking the term feedback per user into account, the 'Semantic Result Optimization' needs to be extended to incorporate the stored user term feedback, resulting in the context specific dimension "user's preference" as described in 4.3.1, Semantic Result Optimization.
In general, the feature would increase the terminologies services' response time but has nevertheless the potential to push the terminology service to its next level.

9.2.5 EHR Medication and Dosage Field

A possible replacement for the existing EHR medication field is shown in figure 9.2. Its basic idea is to have for each individual medication a separate line with two areas instead of the comma separated list.

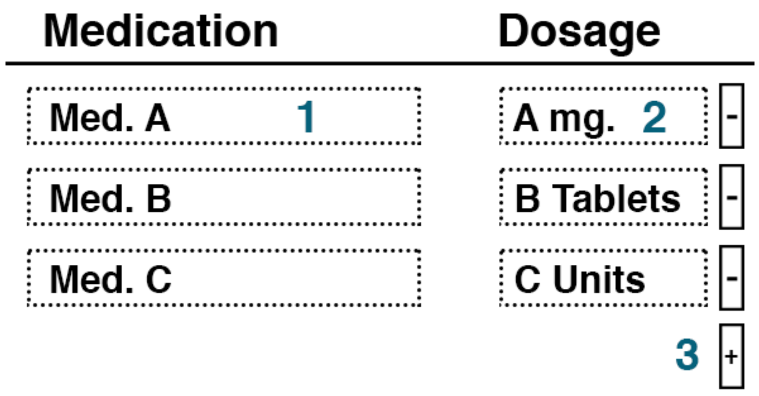


Figure 9.2: Proposal: Medication and Dosage Field

Each lines' first area contains the medication term, facilitating the terminology service. Into the second area the user inputs the dosage of the medication. A user can add more lines on the fly (mark 3). Adding a new dosage term category to the terminology service would allow for dosage autocompletion/autosuggestion. One issue the new medication field solves consists in the separation of medication and dosage, which opens the possibility for the literature service to interpret both values separately. Here, the literature service could be extended to formulate, e.g., a logic 'OR' connected query containing the medication enriched by the existing rules and via a new semantic mapping (depending on the medication) using the 'normalized' dosage concatenated with the medication to form the second part of the OR query. This could increase the usability of the front-end and (by extending the literature service accordingly) result in even more helpful literature.

9.3 Extending the Literature Service

The purpose of this section is the description of possible extensions of the literature service, because parts of the existing service can be re-used, before any of the stated extensions is further planned or implemented, chapter 5, Literature Service, and the services implementation section 6.2, Literature Service, should be considered.

9.3.1 Optimizing Calls to NLM's 'efetch' API

NLM's 'efetch' API, used for literature details retrieval (subsection 6.2.5, Quering PubMed), offers the capability to retrieve details for multiple literature IDs in one call, resulting in fewer calls to the 'efetch' API. Calling the API less, can result in a literature service search speedup. If an implementation takes place the call to the NLM's 'efetch' API needs to be adjusted as well as the literature information extraction relying on XPath queries.

9.3.2 Tune or Replace the Machine Learning Mechanism

As described in subsection 6.2.3, Machine Learning Based Semantic Query Term Identification, a multi-label support vector machine is used to predict query terms for an EHR based on user feedback. Replacing the used linear kernel, could improve prediction outcomes. A functionality not offered by the employed C# library is a directly approach the multi-label classification problem; currently the problem is split in multiple binary classification problems. Aside from building an own multi-label classification solver for a direct approach, external tools offer the necessary capability, two of them are

Matlab
: Zhang and Zhou offer a Matlab code that employs an Artificial Neural Network with a corresponding learning function for multi-label classification problems[57]. The Matlab code is available in the internet at [58], for academic purpose only.

MEKA
: Is an extension of WEKA and written in Java. It offers a set of multi-label classification mechanisms[59].

The integration of Matlab, MEKA or any other tool can be accomplished on different levels, some of them being:

Direct Integration
: If the tool offers a API to C# it can be integrated directly.

Integration without API
: The general idea is to first export all learning data into a representation fitting the

tool and then call the tool on command line level to train the model or calculate predictions. In case of a prediction call, the resulting prediction needs to be imported into the application.

Integration Employing a Wrapper

An external wrapper could be build around the tool like the two above mentioned descriptions. However, instead of invoking the wrapper directly, both application parts communicate via, e.g., a messaging service. By implementing an additional abstraction layer between machine learning component and application, both parts of the application are more decoupled and could evolve at a different pace.

The three implemented transformation steps for the machine learning data have the potential to be re-used should an external machine learning component be employed. In fact, for debugging purpose, all input and outputs of the machine learning are already stored in text files containing JSON formatted matrixes.

9.3.3 EHR Field Mapping

As stated in 6.2.1, Semantic EHR Field Mapping, not all EHR fields are mapped. To ensure the usefulness of the literature search service for unknown EHRs, all EHR fields should be mapped to the literature service's internal representation.

9.3.4 Automatic GUI testing

The view component described in subsection 6.2.6, Highlights of the Literature View Component, is manually tested for desktops and mobile devices by using the Google Chrome browser (in mobile mode). To ensure the end to end functionality of the service for changing browser versions, an automatic testing on integration level should be implemented. All tests should be automatically performed using the latest browsers for both mobile devices as well as for desktops, to ensure the best outcome for the customer's contentment. Automatic end to end testing enables regression tests on integration level.

9.3.5 Sending Relevant Literature via E-Mail

Prior to a MDT meeting, an e-mail containing the helpful literature could be send to the participants. Wether or not the e-mail is send to a certain participant should be configurable by the user in the users' admin panel. The literature service delivers literature objects, they could be transformed into an e-mail fitting format and be sent to the user. There is potential that a user may study the literature thoroughly, hence, the feedback mechanism should be included into the e-mail per literature, as already implemented at the webpage. As the literature feedback service is only available after user login, a security mechanism (e.g., a unique security token) per e-mail and user should be implemented to ensure feedback collection of authorized users only.

9.3.6 Show Voted Literature in User Admin Panel

The literature voting feature comes hand in hand with the probability that a user's vote happens accidentally. For the time being, the voting feature does not take the user ID into account. To enable customized literature suggestions at a later step and displaying already voted literature on user level, the literature feedback service should be extended to incorporate the user ID. In addition, a new user admin panel feature should be implemented with a functionality to delete votes for a certain EHR and literature. As soon as a user vote is changed, the machine learning needs to be informed to accordingly update its model used for predictions.

9.3.7 Personal Literature Suggestions

At the moment literature is shown to the user based on rules operating on mapped EHR fields as well as on user feedback. However, the user feedback is collected and used to predict literature search terms for all users via machine learning. As different users most likely do have different literature needs, a personal literature search term prediction could be implemented. One possibility of achieving this, is to employ a machine learning mechanism per user. As described above a prerequisite for personal literature suggestions is the extension of the literature feedback component to include the user ID.

9.3.8 Crowdsourcing the EHR Literature Feedback

The literature feedback mechanism is available for authorized users only, because of the EHR data privacy requirement. Employing an automatic data anonymisation mechanism on the EHR, the EHR view with corresponding literature and voting functionality could be presented to the public. Prior to the voting, the internet user must choose its skill level and specialization. After refining, the community votes could be used as basis for personal literature predictions via machine learning.

Appendices

Appendix A

Medical Ontologies

Category	Term	The Drug Ontology	National Drug File - Reference Terminology	Human Disease Ontology	Anatomical Entity Ontology	Foundational Model of Anatomy	Uber Anatomy Ontology	Gene Ontology	Ontology of Genes and Genomes	Symptom Ontology	Medical Subject Headings	National Cancer Institute Thesaurus
Medication		x	x								x	x
	Warfarin	x	x								x	x
	NOAC	-	-								x	(x)
	Antiplatelet	-	-								x	x
	Dacarbazine	x	x								x	x
	Ipilimumab	x	-								x	x
Anatomy					x	x	x				x	x
	Right pinna				-	-	-				-	-
	Head				x	-	x				x	x
	Neck				x	-	x				x	x
	Upper Limb				-	-	-				x	x
	Lower Limb				-	-	-				x	x
	Trunk				-	-	x				x	x
	Right Arm				-	-	-				-	x

Category	Term	The Drug Ontology	National Drug File - Reference Terminology	Human Disease Ontology	Anatomical Entity Ontology	Foundational Model of Anatomy	Uber Anatomy Ontology	Gene Ontology	Ontology of Genes and Genomes	Symptom Ontology	Medical Subject Headings	National Cancer Institute Thesaurus
	Left Leg				-	-	-				-	-
	Thorax				-	-	-				x	x
	Abdomen				-	-	x				x	x
	Pelvis				-	-	x				x	x
	Mid Back				-	-	-				-	-
Gene								x	x		x	x
	BRAF							-	x		x	x
	V600E							-	-		-	x
	EGFR							-	-		x	x
Disease				x							x	x
	Moderate Dementia			-							-	-
	Severe Dementia			-							-	(x)
	Primary Melanoma			-							x	(x)
Activity											x	x
	Malignant Melanoma Removed										-	-
	Melanoma In Situ Removed										-	-
	Lesion Excised										-	-
	SNB										(x)	(x)
	CXR										(x)	-
	US										x	x
	Bone Scan										-	x
	Serum LDH										-	-
	Lymphadenectomy										x	x
	CT										x	x
	MRI										x	x
	TETSCAN										-	-

Category	Term	The Drug Ontology	National Drug File - Reference Terminology	Human Disease Ontology	Anatomical Entity Ontology	Foundational Model of Anatomy	Uber Anatomy Ontology	Gene Ontology	Ontology of Genes and Genomes	Symptom Ontology	Medical Subject Headings	National Cancer Institute Thesaurus
	Image Guided Biopsy										x	x
	Ultrasound Study										-	x
	Excision Biopsy										-	x
	Re-Excision Biopsy										-	-
	Wide Excision for Margins										-	-
	Incisional Biopsy										-	x
	Sentinel Node Biopsy										x	x
	FNAC										-	-
	Trucut Biopsy										-	-
	Metastasectomy										x	x
	Immunotherapy										x	x
	Right Axillary Dissection										-	-
	Pet CT Scan										-	(x)
	CT Brain										x	-
Symptom										x	x	x
	Ulceration									(x)	x	x
	Colitis									-	x	x

Table A.1: Medical Terms per Category in Ontologies

Comment on table data:

-	Term was not found in ontology.
x	Term was found in ontology.

(x)

No exact term match found (e.g. plural, multiple matches with different concept,...).

A.1 The Drug Ontology

DrOn contains content curated by the National Library of Medicine and is offered in OWL format as well as in Xlsx format [60] [61]. "The Drug Ontology [...] was originally created to enable comparative effectiveness and health services researchers to query for National Drug Codes that represent products by ingredient, by molecular disposition [...], by therapeutic disposition [...], and by physiological effect [...]" [60].

A.2 National Drug File - Reference Terminology

NDF-RT "is used for modeling drug characteristics including ingredients, chemical structure, dose form, physiologic effect, mechanism of action, pharmacokinetics, and related diseases" [62]. It is produced by the U.S. Department of Veterans Affairs, Veterans Health Administration and part of NLM's UMLS (Unified Medical Language System). On Ontobee it is offered in Xlsx format [63].

A.3 Human Disease Ontology

"The mission the Disease Ontology (DO) is to provide an open source ontology for the integration of biomedical data that is associated with human disease. [. . .] These terms will be linked to well-established, well-adopted terminologies that contain disease and disease-related concepts such as [...] MeSH [...]" [64]. The Human disease ontology can be found as Xlsx file at Ontobee [65].

A.4 Anatomical Entity Ontology

"The AEO is an ontology of anatomical structures that expands CARO, the Common Anatomy Reference Ontology, to about 160 classes[...]. [...] The ontology can be used to classify most organisms as its terms are appropriate for most plant and fungal tissues" [66]. Aside the OWL format the Anatomical Entity Ontology is available as Xlsx file [67].

A.5 Foundational Model of Anatomy

"The Foundational Model of Anatomy ontology makes available anatomical information in symbolic (non-graphical) form to knowledge modelers and other developers of applications for education, clinical medicine, electronic health record, biomedical research and all areas of health care delivery and management" [68]. "The Foundational Model of Anatomy ontology contains approximately 75,000 classes and over 120,000 terms; over 2.1 million relationship instances from over 168 relationship types [...]" [68]. A subset of the ontology can be downloaded as Xlsx file on Ontobee [69].

A.6 Uber Anatomy Ontology

"Uberon is an anatomical ontology that represents body parts, organs and tissues in a variety of animal species, with a focus on vertebrates. It has been constructed to integrate seamlessly with other ontologies, such as [...] the Gene Ontology [...] as well as other anatomical ontologies" [70]. On Ontobee, Uberon is offered in Xlsx format [71].

A.7 Gene Ontology

"The Gene Ontology project provides a controlled vocabulary of terms for describing gene product characteristics and gene product annotation data from GO Consortium members" [72]. "Gene Ontology Consortium data and data products are licensed under the Creative Commons Attribution 4.0 Unported License" [73]. The Gene Ontology can be downloaded as Xlsx file on Ontobee [74].

A.8 Ontology of Genes and Genomes

"The OGG (Ontology of Genes and Genomes) is a biological ontology that represents genes and genomes of various biological organisms" [75]. Its available as Xlsx file on Ontobee [76].

A.9 Symptom Ontology

"The symptom ontology was designed around the guiding concept of a symptom being: 'A perceived change in function, sensation or appearance reported by a patient indicative of a disease'. Understanding the close relationship of Signs and Symptoms, where Signs are the objective observation of an illness, the Symptom Ontology will work to broaden it's scope to capture and document in a more robust manor these two sets of terms. Understanding that at times, the same term may be both a Sign and a Symptom" [77]. The ontology is available both in Xlsx and OWL format [77].

A.10 National Cancer Institute Thesaurus

"NCI Thesaurus (NCIt) provides reference terminology [...]. It covers vocabulary for clinical care, translational and basic research, and public information and administrative activities" [78]. Some features are: Over 100,000 textual definitions and Over 400,000 cross-links between concepts [78]. The NCIt can be download free of charge at [79] but the user has to agree to its terms of use, which can be found here: [80].

A.11 Medical Subject Headings

MeSH is offered by the U.S. National Library of Medicine (NLM) and is their "[...] controlled vocabulary (thesaurus) that gives uniformity and consistency to the indexing and cataloging of biomedical literature" [81], [82]. It is "[...] used for indexing articles of PubMed" [82] and is "arranged in a hierarchical manner called the MeSH Tree Structures." [81] and updated annually around mid November [83]. MeSH can be obtained free-of-charge without a license after agreeing to a Memorandum of Understanding and completing a registration. The Memorandum of Understanding includes restrictions a user has to abide. Those include the exclusion of any proprietary rights for users, the U.S. assumes no legal liability and others. All details can be found on the corresponding NLM webpage: [83]. MeSH offers 16 different subtrees, the complete list can be found at [51].

Appendix B

Medical Literature Queries

B.1 Query of 3 EHR fields with Comma as Separator

```
warfarin ,male , dementia
```

Listing B.1: Query 1, Comma as Separator

B.2 Query of 1 EHR field with Whitespace as Separator

```
invasive malignant melanoma recently removed
```

Listing B.2: Query 2, Whitespace as Separator

B.3 Query of 12 EHR fields with Comma as Seperator

```
Male , Full time Residential Care , dementia , Warfarin ,NOACs, Antiplatelet
    Agents , invasive malignant melanoma recently removed , Primary Melanoma site
    Head and neck , Breslow thickness 4.0 mm, Mitosis 3mm^2 , Ulceration , pT1a
```

Listing B.3: Query 3, Whitespace for Multiword Terms and Comma as Separator

B.4 Query of 12 EHR fields with Whitespace as Separator

```
Male Full time Residential Care dementia Warfarin NOACs Antiplatelet Agents
    invasive malignant melanoma recently removed Primary Melanoma site Head
    and neck Breslow Ulceration pT1a
```

Listing B.4: Query 4, Whitespace as Separators

B.5 Query of 12 EHR fields with "or" as Separator Variant 1

```
((((((((((male) OR full time residential care) OR dementia) OR warfarin) OR
    noac) OR antiplatelet agents) OR invasive malignant melanoma recently
    removed) OR (primary melanoma site head and neck)) OR breslow thickness
    4.0 mm) OR mitosis 3mm^2) OR ulceration) OR pT1a
```

Listing B.5: Query 5, Brackets for Multiword Terms and "or" as Separator - Variant 1

B.6 Query of 12 EHR fields with "or" as Separator Variant 2

```
(male) OR (full time residential care) OR (dementia) OR (warfarin) OR (noac)
    OR (antiplatelet agents) OR (invasive malignant melanoma recently
    removed) OR (primary melanoma site head and neck) OR (breslow thickness
    4.0 mm) OR (mitosis 3mm^2) OR (ulceration) OR (pT1a)
```

Listing B.6: Query 6, Brackets for Multiword Terms and "or" as Separator - Variant 2

B.7 Query of 15 EHR fields with "or" as Separator

```
(male) OR (full time residential care) OR (dementia) OR (warfarin) OR (noac)
    OR (antiplatelet agents) OR (invasive malignant melanoma recently
    removed) OR (primary melanoma site head and neck) OR (breslow thickness
    4.0 mm) OR (mitosis 3mm^2) OR (ulceration) OR (pT1a) OR (Bone scan) OR
    (CT Neck) OR (Positive BRAF Status)
```

Listing B.7: Query 7, Brackets for Multiword Terms and "or" as Separator

Appendix C

Medical Literature Search Engines

Search Engine	Base Data	Usage Licence Type (not the documents)	Type	API	Remark	URL
PubMed	PubMed/MEDLINE	public domain	service	y	-	http://www.ncbi.nlm.nih.gov/pubmed/advanced
Textpresso	PubMed	noncommercial	platfrom, service	-		http://brahma.textpresso.org/cancer
HubMed	PubMed/MEDLINE	not known	service	?		http://git.macropus.org/hubmed
Medlineplus	PubMed/MEDLINE and others	public domain	service	y		http://www.nlm.nih.gov/medlineplus
Quertle	MEDLINE and others	free, no robots	service	?		http://www.quertle.info
ScienceDirect	˜2 500 journals more than 30 000 books	free	service	?		http://www.sciencedirect.com
Medscape	MEDLINE and others	free, advertisements	service	?		http://www.medscape.com
BibliMed	MEDLINE	free, subscription	service	?	no about page	http://www.biblimed.com
ASTIS	80 000 records of describing publications and research projects	not known	service	?		http://www.aina.ucalgary.ca/scripts/minisa.dll?HOME
DOAJ	10 535 Journals; 1 903 095 Articles	open access	service	y		https://doaj.org
Google Scholar	Various databases	google	service	?	query is cut	https://scholar.google.com
HighWire	Publications: 3546 - Articles: 7 100 000	not known	service	?	query is cut	http://home.highwire.org
GoPubMed	PubMed/MEDLINE and others	not known	service	?		http://www.gopubmed.org/web/gopubmed/
POLS	not known	open access	service	?		https://www.plos.org
LIVIVO	PubMed/MEDLINE and others	not known	service	?		http://www.zbmed.de/recherchieren/livivo/
BioMed Central	not known	open access, advertisements	service	y		http://www.biomedcentral.com
Pubget	PubMed an others	not known	service	y	copyright clearance center	http://pubget.com

Table C.1: Literature Search Engine Overview

C.1 PubMed

This is provieded by the US National Library of Medicine. "PubMed comprises more than 24 million citations for biomedical literature from MEDLINE, life science journals, and online books. Citations may include links to full-text content from PubMed Central and publisher web sites" [84], [85]. While quering PubMed the MeSH tree structure is used. The information found in PubMed comes in two distinct flavors: a free and a non-free set. Hence not only the publications abstract but the complete full text can be retrieved without any price attached [86]. PubMed disposes of more than 24 million articles [87]. 3.4 million of them are free articles [88] the rest are non-free. Different ways to access the sets are provided: aside from a search engine both as webpage and API, it is even possible to download all of the free articles in compressed files. If a direct download is chosen, two formats are offered: XML and plain text. It is possible to retrieve the complete free set of publications by downloading rather large but view compressed files only [89]. The Zip file's size for articles A-B in XML format of the free set is larger than 3GB.

C.2 Textpresso

Textpresso is an information extracting and processing package for biological literature [90], [91]. "The Textpresso project serves the biological and biomedical research community by providing:
Full text literature searches of model organism research and subject-specific articles at individual sites[...]. Text classification and mining of biomedical literature for database curation[...]. Linking biological entities in PDF and online journal articles to online databases[...]" [91]. The software "[...]is part of WormBase at the California Institute of Technology, California" [90]. The software is free to use for noncommercial purposes.

C.3 HubMed

"HubMed (bearing the tagline 'PubMed rewired') is an alternative interface to the PubMed database. Created by script-savvy programmer Alf Eaton, HubMed pulls information directly from PubMed via National Center for Biotechnology Information's (NCBI) Entrez Programming Utilities. In addition to serving as an excellent demonstration of the power of XML and the semantic Web, HubMed offers innovative ways to browse and explore the literature" [92], [93].

C.4 Medlineplus

"MedlinePlus is the National Institutes of Healthś Web site for patients and their families and friends. Produced by the National Library of Medicine, the world's largest medical library, it brings you information about diseases, conditions, and wellness issues in language you can understand. MedlinePlus offers reliable, up-to-date health information, anytime, anywhere, for free" [94]. Medlineplus contains links to preformulated searches of the Medline/PubMed database as well as information from other sources [95]. An API is offerd [96].

C.5 Quertle

This is owned by Quertle LLC. "Quertle uses MEDLINE/PubMed® as provided by the US National Library of Medicine (NLM) [...] and full-text documents from BioMed Central and Open Access articles from PubMed Central. In addition, Quertle covers the NIH RePORTER database of grant applications and the National Library of Medicine's TOXLINE database of biochemical, pharmacological, physiological, and toxicological effects of drugs and other chemicals. Quertle also searches News (as reported by FierceMarkets Life Sciences and Health Care)[...]" [97]. The usage of Quertle is free, no robots are allowed [98].

C.6 ScienceDirect

The publishing house Elsevier B.V. own this product [99]. According to the webpage it "... is a leading full-text scientific database offering journal articles and book chapters from nearly 2,500 journals and more than 30,000 books" [100].

C.7 Medscape

This product belogs to WebMD LLC [101]. "Medscape from WebMD offers specialists, primary care physicians, and other health professionals the Web's most robust and integrated medical information and educational tools. After a simple, 1-time, free registration, Medscape from WebMD automatically delivers [...] a personalized specialty site [...]" [102]. The information displayed on medscape comes from Medline and other sources [102].

C.8 BibliMed

"BibliMed is a search interface based on Medline biomedical database including more than 23 million scientific articles [...] From Medline database" [103]. At the time of writing there was no about page but a contact section, showing an E-Mail address only [103]. According to a DNS 'who is query' the domain is registerd at http://www.gandi.net an entity in Paris (France) [104]. The Interface is offered in French as well. On BibliMed's Twitter page, four out of five followers are writing in French [105].

C.9 Arctic Science and Technology Information System

Is availabel at [106]. "The Arctic Science and Technology Information System (ASTIS) database contains 80,000 records describing publications and research projects about northern Canada. ASTIS, a project of the Arctic Institute of North America at the University of Calgary, also maintains subset databases about specific regions, subjects and projects." [107]. "ASTIS covers all aspects of northern Canada, including [...] the life sciences [...]" [108].

C.10 Directory of Open Access Journals

DOAJ was founded as a project of the Lund University Libraries (Sweden) it is nowadays maintain and develop by is4oa (Infrastructure Services for Open Access C.I.C.) [109]. "The aim of the DOAJ is to increase the visibility and ease of use of open access scientific and scholarly journals, thereby promoting their increased usage and impact. The DOAJ aims to be comprehensive and cover all open access scientific and scholarly journals that use a quality control system to guarantee the content. In short, the DOAJ aims to be the one-stop shop for users of open access journals" [110]. DOAJ offers an API and searches in 10,510 journals where 6,308 can be searched on article level from 134 countries resulting in 1,897,516 articles [111].

C.11 Google Scholar

"Google Scholar provides a simple way to broadly search for scholarly literature. From one place, you can search across many disciplines and sources: articles, theses, books, abstracts and court opinions, from academic publishers, professional societies, online repositories, universities and other web sites. Google Scholar helps you find relevant work across the world of scholarly research" [112].

C.12 HighWire

This tool offered by HighWire Press, Inc. According to its website it is "A leading ePublishing platform, HighWire Press partners with independent scholarly publishers, societies, associations, and university presses to facilitate the digital dissemination of more than 3000 journals, books, reference works, and proceedings." "HighWire has been affiliated with Stanford University since 1995 [...]" [113].

C.13 GoPubMed

"The fundamental difference between GoPubMed's® semantic search technology and traditional search engines such as PubMed or Google is the use of background knowledge. Semantic algorithms connect text – abstracts from the MEDLINE database – to background knowledge in the form of semantic networks of concept categories, also called ontologies

or knowledge base [...]. The knowledge behind GoPubMed® consists of in total 48 million concepts – the majority of 47 million come from authors, 45,000 from the GO and MeSH ontologies (plus about 150,000 synonyms), 62,000 from proteins, 23,000 from journals, 13,000 from geolocations, and 35,000 from time-related concepts" [114].
GoPubMed is based on the software platform “Enterprise Semantic Intelligence®” a product of Transinsight GmbH [115]. "The firm works in close collaboration with the Dresden University of Technology" in Germany [115].

C.14 Public Library of Science

"PLOS (Public Library of Science) is a nonprofit publisher and advocacy organization founded to accelerate progress in science and medicine by leading a transformation in research communication" [116].

C.15 LIVIVO

"LIVIVO includes a complete PubMed-search and other important databases in life sciences such as AGRICOLA or AGRIS." [117] While AGRICOLA stands for AGRICultural OnLine Access.[118] AGRIS is the database of the International System for Agricultural Science and Technology. [119] "[...] The search terms are automatically linguistically enriched and semantically linked by vocabularies that were specially designed and adapted for life sciences (using MeSH for medical sciences [...])" [117]. The software is operated by ZB MED - Leibniz Information Centre for Life Sciences being part of the German National Library of Medicine its Supervisory authority is the North Rhine Westphalian Ministry of Innovation, Science, Research and Technology [120], [117].

C.16 BioMed Central

"BioMed Central is an STM (Science, Technology and Medicine) publisher of 279 peer-reviewed open access journals" [121]. It is Part of Springer Science+Business Media [121]. " All original research articles published by BioMed Central are made freely accessible online immediately upon publication" [121]. An API as well as a complete download is offered [122].

C.17 Pubget

"Pubget is the search engine for life science PDFs. Pubget makes scientific research easier by simplifying the process of finding, managing and analyzing scientific papers. The core solution at www.pubget.com provides article-level tools making content discovery, access and copyright management much easier for the user. [...] Pubget is a Copyright Clearance Center company. Together our products provide clients with complete information and media solutions" [123].

Appendix D

Implementation Source Code

D.1 Solr - schema.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema name="TermStore" version="1.5">
  <fields>
    <field name="_version_" type="long" indexed="true" stored="true"/>

    <!-- Term class -->
    <field name="TermId" type="string" indexed="true" stored="true"
        required="true" multiValued="false" />
    <field name="SourceId" type="string" indexed="true" stored="true"
        required="false" multiValued="false" />
    <field name="Label" type="string" indexed="true" stored="true"
        required="true" />
    <field name="Broader" type="string" indexed="false" stored="true"
        multiValued="true" />
    <field name="Synonyms" type="string" indexed="true" stored="true"
        multiValued="true" />
    <field name="Definition" type="string" indexed="true" stored="true"
        multiValued="false" />
    <field name="Category" type="string" indexed="true" stored="true"
        multiValued="false" />

    <field name="autocomplete_text" type="text_suggest" indexed="true"
        stored="false" multiValued="true" omitNorms="false"
        omitTermFreqAndPositions="false" />
  </fields>
    <uniqueKey>TermId</uniqueKey>

    <copyField source="Label" dest="autocomplete_text" />
    <copyField source="Synonyms" dest="autocomplete_text" />

  <types>
    <fieldType name="string" class="solr.TextField"
        positionIncrementGap="100" />
    <fieldType name="text_suggest" class="solr.TextField"
        positionIncrementGap="100">
      <analyzer type="index">
        <tokenizer class="solr.WhitespaceTokenizerFactory" />
        <filter class="solr.LowerCaseFilterFactory" />
        <filter class="solr.EdgeNGramFilterFactory" minGramSize="1"
            maxGramSize="100"/>
      </analyzer>

      <analyzer type="query">
        <tokenizer class="solr.WhitespaceTokenizerFactory" />
          <filter class="solr.LowerCaseFilterFactory" />
      </analyzer>
    </fieldType>
  </types>
</schema>
```

Listing D.1: Solr - schema.xml

D.2 Solr - Term Insert Document

```
public class SolrTermInsertDocument : IEquatable<SolrTermInsertDocument>
{
  [SolrUniqueKey("TermId")]
  public string TermId { get; set; }
  [SolrField("SourceId")]
  public string SourceId { get; set; }
  [SolrField("Label")]
  public string Label { get; set; }
  [SolrField("Broader")]
  public ICollection<string> Broader { get; set; }
  [SolrField("Synonyms")]
  public ICollection<string> Synonyms { get; set; }
  [SolrField("Definition")]
  public string Definition { get; set; }
  [SolrField("Category")]
  public string Category { get; set; }

  public override bool Equals(object other)
  {
    return other != null && other is SolrTermInsertDocument &&
        other.GetHashCode() == this.GetHashCode();
  }
  public bool Equals(SolrTermInsertDocument other)
  {
    return other != null && other.GetHashCode() == this.GetHashCode();
  }
  public override int GetHashCode()
  {
    return 42 + this.SourceId.GetHashCode();
  }
}
```

Listing D.2: Solr - Term Insert Document

D.3 Solr - Term Query Document

```
public class SolrTermQueryDocument : IEquatable<SolrTermQueryDocument>
{
  [SolrUniqueKey("TermId")]
  public string TermId { get; set; }
  [SolrField("SourceId")]
  public string SourceId { get; set; }
  [SolrField("Label")]
  public string Label { get; set; }
  [SolrField("Broader")]
  public ICollection<string> Broader { get; set; }
  [SolrField("Synonyms")]
  public ICollection<string> Synonyms { get; set; }
  [SolrField("Definition")]
  public string Definition { get; set; }
  [SolrField("Category")]
  public string Category { get; set; }
  [SolrField("autocomplete_text")]
  public string AutocompleteText { get; set; }
  [SolrField("score")]
  public double? Score { get; set; }

  public override bool Equals(object other)
  {
    return other != null && other is SolrTermQueryDocument &&
        other.GetHashCode() == this.GetHashCode();
  }
  public bool Equals(SolrTermQueryDocument other)
  {
    return other != null && other.GetHashCode() == this.GetHashCode();
  }
  public override int GetHashCode()
  {
    return 42 + this.SourceId.GetHashCode();
  }
}
```

Listing D.3: Solr - Term Query Document

D.4 PubMed XML Parser Based on XPath

```
public string perfromXpath(string XpathExpression, string GroupSeperator = ",
    ", string InnerGroupSeperator = " ")
{
  StringBuilder ReturnValue = new StringBuilder();

  int GroupCount = Regex.Matches(XpathExpression, "\\|").Count;
  if (GroupCount == 0)
  {
    XmlNode Node = XmlDocument.SelectSingleNode(XpathExpression);
      if (Node != null) ReturnValue.Append(Node.InnerXml);
  }
  else
  {
    XmlNodeList Nodes = XmlDocument.SelectNodes(XpathExpression);
    if (Nodes != null)
    {
      for (int j = 0; j < Nodes.Count; j++)
      {
        ReturnValue.Append(Nodes.Item(j).InnerXml);
        // not the very first element
        if (j != 0
        // every time a group is complete, +1 caused by groups of one having
            two columns
        && (j + 1) % (GroupCount + 1) == 0
        // not the very last element
        && j != Nodes.Count - 1)
        {
          ReturnValue.Append(GroupSeperator);
        }
        else
        {
          ReturnValue.Append(InnerGroupSeperator);
        }
      }
    }
  }
  return ReturnValue.ToString();
}
```

Listing D.4: PubMed XML Parser Based on XPath

Appendix E

Test Run: Semantic Query Term Identification

E.1 Search Terms Identified by Rules

Using one of the demo EHRs, the rules formulate the following query shown in listing E.1

```
((ipilimumab efficacy) OR (ipilimumab safety)) AND (stage 0) AND (melanoma
    [mh:noexp])
```

Listing E.1: PubMed Query Created by Rules

The result from PubMed are 14 literatures.

E.2 Search Terms from the Multi-Label SVM

Using a ML SVM trained with all available test data (EHR and known useful literature), the query shown in listing E.2 is submitted.

```
(melanoma) AND (lymph node) AND (Immunotherapy) AND (peptide) AND
    (glycoprotein) AND (Combined Modality Therapy) AND (external ear) AND
    (metastatic melanoma) AND (Interferon Alfa-2b) AND (Sulfonamides) AND
    (cancer) AND (indoles) AND (Proteins) AND (Ear)
```

Listing E.2: PubMed Query Created by Machine Learning - To Specific Query

The result from PubMed are zero literatures. Because EHR corresponding helpful literature is known, data like the following statistics are automatically computed by the program (a feature of the employed 'Accord.Net' library) and stored in JSON format to disk:

Samples	=39	Actual Positives	=5	Actual Negatives	=34		
Predicted Positives	=14	True Positives	=2	False Positives	=12	Precision ($\frac{\text{True Positives}}{\text{Predicted Positives}}$)	$=\frac{1}{7}$
Predicted Negatives	=34	False Negatives	=3	True Negatives	=22		
Recall ($\frac{\text{True Positives}}{\text{Actual Positives}}$)	$=\frac{2}{5}$						

Table E.1: Multi-Label SVM - Confusion Matrix

E.3 Post Processed Search Terms from the Multi-Label SVM

Because the first query from the machine learning did not turn up any results, the lower ranked query terms are removed till a search returns literature. Listing E.3 shows the query.

```
(melanoma) AND (lymph node) AND (Immunotherapy) AND (peptide) AND
    (glycoprotein) AND (Combined Modality Therapy)
```

Listing E.3: PubMed Query Created by Machine Learning

The result from PubMed are 10 literatures.
Both results the 10 literatures from the machine learning and the 14 literatures from the rules are presented to the user.

Bibliography

[1] (7/13/2015). SAGE-CARE - EU - CORDIS, [Online]. Available: `http://cordis.europa.eu/project/rcn/194165_en.html`.

[2] K. K. Jain, *Textbook of Personalized Medicine*, 2nd Edition. Springer Humana Press, 2015.

[3] (7/20/2015). U.S. Food and Drug Administration - Personalized Medicine, [Online]. Available: `http://www.fda.gov/scienceresearch/specialtopics/personalizedmedicine/default.htm`.

[4] (7/20/2015). European Alliance for Personalised Medicine - Report from Irish Presidency Conference March 20th/21st 2013, [Online]. Available: `http://euapm.eu/wp-content/uploads/2012/07/EAPM-REPORT-on-Innovation-and-Patient-Access-to-Personalised-Medicine.pdf`.

[5] (7/13/2015). NSilico - Contact, [Online]. Available: `http://www.nsilico.com/Contact`.

[6] (7/13/2015). Linked in - NSilico, [Online]. Available: `https://www.linkedin.com/company/nsilico`.

[7] (7/13/2015). NSilico - About, [Online]. Available: `http://www.nsilico.com/About`.

[8] (7/13/2015). NSilico - Simplicity, [Online]. Available: `http://www.nsilico.com/Simplicity`.

[9] (7/13/2015). NSilico - Simplicity MDT, [Online]. Available: `http://www.nsilico.com/SimplicityMDT`.

[10] C. Grosan and A. Abraham, *Intelligent Systems A Modern Approach*. Springer, 2011, vol. 17.

[11] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, 2002.

[12] D. Floreano and C. Mattiussi, *Bio-Inspired Artificial Intelligence*. The MIT Press, 2008.

[13] (8/18/2015). Statistical foundations of machine learning - 7.4.2 Support vector machines, [Online]. Available: `https://www.otexts.org/1585`.

[14] "Kernels," in *Encyclopedia of Machine Learning*, C. Sammut and G. Webb, Eds., Springer US, 2010.

[15] (8/19/2015). An Idiot's guide to Support vector machines - Ans: polar coordinates, [Online]. Available: `http://www.cs.ucf.edu/courses/cap6412/fall2009/papers/Berwick2003.pdf`.

[16] O. Maimon and L. Rokach, Eds., *Data Mining and Knowledge Discovery Handbook*, 2. New York: Springer, 2010.

[17] S. Abe, *Support Vector Machines for Pattern Classification (Advances in Pattern Recognition)*. Secaucus, NJ, USA: Springer-Verlag New York, 2005.

[18] (8/20/2015). A Comparison of Multiclass SVM Methods, [Online]. Available: `http://courses.media.mit.edu/2006fall/mas622j/Projects/aisen-project/`.

[19] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY, USA: Springer-Verlag New York, 1995.

[20] P. Øhrstrøm, J. Andersen, and H. Schärfe., "What has happened to ontology," *Conceptual Structures: Common Semantics for Sharing Knowledge, 13th International Conference on Conceptual Structures*, vol. 3596, 425 pp. 2005.

[21] N. Guarino, D. Oberle, and S. Staab, *Handbook on Ontologies*. Springer, 2009.

[22] J. Busse, B. Humm, C. Lübbert, F. Moelter, A. Reibold, M. Rewald, V. Schlüter, B. Seiler, E. Tegtmeier, and T. Zeh, "Was bedeutet eigentlich ontologie?," *Informatik Spektrum*, vol. 37, no. 4, 286 pp, 2014.

[23] (7/20/2015). jfsowa.com - Ontology, [Online]. Available: `http://www.jfsowa.com/ontology/`.

[24] T. Pellegrini and A. Blumauer, *Semantic Web Wege zur vernetzten Wissensgesellschaft*. Springer, 2006.

[25] L. Liu and M. T. Özsu, Eds., *Encyclopedia of Database Systems*. Springer US, 2009.

[26] (7/16/2015). Wikipedia - List of academic databases and search engines, [Online]. Available: `http://en.wikipedia.org/wiki/List_of_academic_databases_and_search_engines`.

[27] S. Raychaudhuri, *Computational Text Analysis for Functional Genomics and Bioinformatics*. Oxford University Press, 2006.

[28] (7/29/2015). U.S. National Library of Medicine - About, [Online]. Available: `http://www.nlm.nih.gov/about/index.html`.

[29] (7/29/2015). U.S. National Library of Medicine - Databases, Resources and APIs, [Online]. Available: `https://wwwcf2.nlm.nih.gov/nlm_eresources/eresources/search_database.cfm`.

[30] (7/29/2015). National Center for Biotechnology Information - Entrez Programming Utilities Help, [Online]. Available: http://www.ncbi.nlm.nih.gov/books/NBK25501/.

[31] (7/29/2015). National Center for Biotechnology Information - Sample Applications of the E-utilities, [Online]. Available: http://www.ncbi.nlm.nih.gov/books/NBK25498/.

[32] (8/18/2015). A General Introduction to the E-utilities - Entrez Programming Utilities Help, [Online]. Available: http://www.ncbi.nlm.nih.gov/books/NBK25497/.

[33] A. Mosa and I. Yoo, "Association mining of search tags in pubmed search sessions," in *Bioinformatics and Biomedicine (BIBM), 2014 IEEE International Conference on*, Nov. 2014, pp. 56–61.

[34] A. Serafini, *Apache Solr Beginner's Guide*. Packt Publishing, 2013.

[35] (7/15/2015). Apache Lucene, [Online]. Available: http://lucene.apache.org.

[36] R. Kuć, *Apache Solr 4 Cookbook*. Packt Publishing, 2013.

[37] (7/15/2015). Eclipse Foundation - Jetty, [Online]. Available: http://www.eclipse.org/jetty/.

[38] (7/15/2015). Oracle - The Java EE Tutorial - Packaging Web Archives, [Online]. Available: https://docs.oracle.com/javaee/7/tutorial/packaging003.htm.

[39] (7/15/2015). Solr Wiki - Tomcat, [Online]. Available: https://wiki.apache.org/solr/SolrTomcat.

[40] (7/15/2015). World Wide Web Consortium (W3C) - Cross-Origin Resource Sharing, [Online]. Available: http://www.w3.org/TR/cors/.

[41] (7/15/2015). Apache Solr Reference Guide - Dynamic Fields, [Online]. Available: https://cwiki.apache.org/confluence/display/solr/Dynamic+Fields.

[42] (7/15/2015). Apache Solr Reference Guide - Copying Fields, [Online]. Available: https://cwiki.apache.org/confluence/display/solr/Copying+Fields.

[43] (7/15/2015). Solr Wiki - Analyzers, Tokenizers, TokenFilters, [Online]. Available: https://wiki.apache.org/solr/AnalyzersTokenizersTokenFilters.

[44] (7/15/2015). Solr Wiki - schema.xml, [Online]. Available: https://wiki.apache.org/solr/SchemaXml.

[45] (7/15/2015). Solr Wiki - config.xml, [Online]. Available: https://wiki.apache.org/solr/SolrConfigXml.

[46] (7/15/2015). stackoverflow - how to delete all data from solr and hbase, [Online]. Available: http://stackoverflow.com/questions/7722508/how-to-delete-all-data-from-solr-and-hbase.

[47] (7/13/2015). Melanoma Institute Australia Multidisciplinary Team, [Online]. Available: http://www.canrefer.org.au/service/447.

[48] J. Gula, *Personalized Medicine - Zulassungsarbeit zum Dipôlme d'Ingénieur im Cnam-Studiengang*. Hochschule Darmstadt - Fachbereich Informatik, Wintersemester 2014/15.

[49] Z. Xiang, C. Mungall, A. Ruttenberg, and Y. He, "Ontobee: a linked data server and browser for ontology terms.," in *ICBO*, O. Bodenreider, M. E. Martone, and A. Ruttenberg, Eds., ser. CEUR Workshop Proceedings, vol. 833, CEUR-WS.org, 2011.

[50] (5/11/2015). Cancer Research UK, [Online]. Available: http://www.cancerresearchuk.org/about-cancer/type/melanoma/treatment/stages-of-melanoma#breslow.

[51] (7/29/2015). U.S. National Library of Medicine - MeSH Tree Structures - 2015, [Online]. Available: https://www.nlm.nih.gov/mesh/trees.html.

[52] (7/29/2015). Ontobee- Welcome to Ontobee, [Online]. Available: http://www.ontobee.org.

[53] (5/04/2015). PubMed Help - Ages, [Online]. Available: http://www.ncbi.nlm.nih.gov/books/NBK3827/.

[54] T. Deuschel, C. Greppmeier, B. G. Humm, and W. Stille, "Semantically faceted navigation with topic pies," *Proceedings of the 10th International Conference on Semantic Systems (SEMANTiCS 2014)*, 2014.

[55] Y. Feng, L. Zheng, L. Song, H. Duan, and N. Deng, "Quantitative measurement of clinic-genomic association for colorectal cancer using literature mining and google-distance algorithm," in *Biomedical Engineering and Informatics (BMEI), 2014 7th International Conference on*, Oct. 2014, pp. 734–739.

[56] S. Pitigala, C. Li, and S. Seo, "A comparative study of text classification approaches for personalized retrieval in pubmed," in *Bioinformatics and Biomedicine Workshops (BIBMW), 2011 IEEE International Conference on*, Nov. 2011, pp. 919–921.

[57] M.-L. Zhang and Z.-H. Zhou, "Multilabel neural networks with applications to functional genomics and text categorization," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 18, no. 10, pp. 1338–1351, Oct. 2006.

[58] (8/18/2015). Resources - Codes for multi-label learning, [Online]. Available: http://cse.seu.edu.cn/people/zhangml/Resources.htm.

[59] (8/18/2015). MEKA: A Multi-label Extension to WEKA, [Online]. Available: http://meka.sourceforge.net.

[60] (7/20/2015). Drug Ontology Home, [Online]. Available: https://ontology.atlassian.net/wiki/display/DRON/Drug+Ontology+Home.

[61] (7/20/2015). Ontobee - The Drug Ontology, [Online]. Available: http://www.ontobee.org/browser/index.php?o=DRON.

[62] (7/20/2015). U.S. National Library of Medicine - National Drug File - Reference Terminology, [Online]. Available: http://www.nlm.nih.gov/research/umls/sourcereleasedocs/current/NDFRT/.

[63] (7/20/2015). Ontobee - National Drug File - Reference Terminology, [Online]. Available: http://www.ontobee.org/browser/index.php?o=DRON.

[64] (7/20/2015). Human disease ontology - About, [Online]. Available: http://disease-ontology.org/about/.

[65] (7/20/2015). Ontobee - Human disease ontology, [Online]. Available: http://www.ontobee.org/browser/index.php?o=DRON.

[66] (7/20/2015). Ontology of Anatomical Entities - Main Page, [Online]. Available: http://wiki.obofoundry.org/wiki/index.php/AEO:Main_Page.

[67] (7/20/2015). Ontobee - Ontology of Anatomical Entities, [Online]. Available: http://www.ontobee.org/browser/index.php?o=AEO.

[68] (7/20/2015). Foundational Model of Anatomy - About, [Online]. Available: http://sig.biostr.washington.edu/projects/fm/AboutFM.html.

[69] (7/20/2015). Ontobee - Foundational Model of Anatomy (subset), [Online]. Available: http://www.ontobee.org/browser/index.php?o=FMA.

[70] (7/20/2015). Uberon - About, [Online]. Available: http://uberon.github.io/about.html.

[71] (7/20/2015). Ontobee - Uberon, [Online]. Available: http://www.ontobee.org/browser/index.php?o=UBERON.

[72] (7/20/2015). Gene Ontology - About, [Online]. Available: http://geneontology.org/page/about.

[73] (7/20/2015). Gene Ontology - Use and License, [Online]. Available: http://geneontology.org/page/use-and-license.

[74] (7/20/2015). Ontobee - Gene Ontology, [Online]. Available: http://www.ontobee.org/browser/index.php?o=GO.

[75] (7/20/2015). OGG (Ontology of Genes and Genomes), [Online]. Available: https://code.google.com/p/ogg/.

[76] (7/20/2015). Ontobee - Ontology of Genes and Genomes, [Online]. Available: http://www.ontobee.org/browser/index.php?o=OGG.

[77] (7/20/2015). Ontobee - Symptom Ontology, [Online]. Available: http://www.ontobee.org/browser/index.php?o=SYMP.

[78] (7/20/2015). National Cancer Institut - NCI thesaurus, [Online]. Available: https://ncit.nci.nih.gov/ncitbrowser/.

[79] (7/20/2015). National Cancer Institut - NCI thesaurus download, [Online]. Available: http://evs.nci.nih.gov/ftp1/NCI_Thesaurus/.

[80] (7/20/2015). National Cancer Institut - NCI thesaurus terms of use, [Online]. Available: http://evs.nci.nih.gov/ftp1/NCI_Thesaurus/ThesaurusTermsofUse.htm.

[81] (7/16/2015). U.S. National Library of Medicine - What is MeSH, [Online]. Available: http://www.nlm.nih.gov/bsd/disted/meshtutorial/introduction/02.html.

[82] (5/11/2015). U.S. National Library of Medicine - MeSH, [Online]. Available: http://www.ncbi.nlm.nih.gov/mesh.

[83] (7/16/2015). U.S. National Library of Medicine - MeSH® Memorandum of Understanding, [Online]. Available: http://www.nlm.nih.gov/mesh/termscon.html.

[84] (5/04/2015). PubMed - advanced search, [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/advanced.

[85] (5/04/2015). PubMed Description, [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed.

[86] (5/11/2015). PubMed - Open Access Subset, [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/tools/openftlist/.

[87] (5/11/2015). PubMed, [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/.

[88] (5/11/2015). PubMed - access, [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/.

[89] (5/11/2015). PubMed - FTP access, [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/tools/ftp/.

[90] (5/11/2015). Textpresso, [Online]. Available: http://textpresso.yeastgenome.org/textpresso/.

[91] (5/11/2015). Textpresso - Home, [Online]. Available: http://www.textpresso.org.

[92] (5/11/2015). HubMed - Purpose and general description, [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1773043/.

[93] (5/04/2015). HubMed, [Online]. Available: http://git.macropus.org/hubmed/.

[94] (5/04/2015). MedLinePlus About, [Online]. Available: http://www.nlm.nih.gov/medlineplus/faq/difference.html.

[95] (5/04/2015). MedLinePlus difference to PubMed, [Online]. Available: http://www.nlm.nih.gov/medlineplus/aboutmedlineplus.html.

[96] (5/04/2015). MedLinePlus Webservices, [Online]. Available: http://www.nlm.nih.gov/medlineplus/webservices.html.

[97] (5/04/2015). Quertle About, [Online]. Available: http://www.quertle.info/pages/about.shtml.

[98] (5/04/2015). Quertle Terms, [Online]. Available: http://www.quertle.info/pages/terms.shtml.

[99] (5/04/2015). Elsevier, [Online]. Available: http://www.elsevier.com.

[100] (5/04/2015). ScienceDirect, [Online]. Available: http://www.sciencedirect.com.

[101] (5/04/2015). Medscape, [Online]. Available: http://www.medscape.com.

[102] (5/04/2015). Medscape - About, [Online]. Available: http://www.medscape.com/public/about.

[103] (5/04/2015). Biblimed, [Online]. Available: http://www.biblimed.com.

[104] (5/04/2015). nic.com - Who Is, [Online]. Available: http://nic.com/.

[105] (5/04/2015). Twitter Biblimed, [Online]. Available: https://twitter.com/biblimed.

[106] (5/05/2015). Arctic Science and Technology Information System, [Online]. Available: http://www.aina.ucalgary.ca/scripts/minisa.dll?HOME.

[107] (5/04/2015). Arctic Science and Technology Information System - About, [Online]. Available: http://www.aina.ucalgary.ca/astis/.

[108] (5/04/2015). Arctic Science and Technology Information System - What's In ASTIS, [Online]. Available: http://www.aina.ucalgary.ca/scripts/minisa.dll/14/astis/contentse.htm?GET.

[109] (5/04/2015). Infrastructure Services for Open Access C.I.C., [Online]. Available: http://is4oa.org/services/doaj/.

[110] (5/04/2015). Directory of Open Access Journals - About, [Online]. Available: https://doaj.org/faq.

[111] (5/04/2015). Directory of Open Access Journals - Features, [Online]. Available: https://doaj.org/features.

[112] (5/06/2015). Google Scholar, [Online]. Available: https://scholar.google.com/intl/en/scholar/about.html.

[113] (5/11/2015). HighWire - About, [Online]. Available: http://home.highwire.org/about-us.

[114] (5/11/2015). GoPubMed - Help, [Online]. Available: http://help.gopubmed.com.

[115] (5/11/2015). Transinsight - About, [Online]. Available: http://transinsight.com/about-transinsight/.

[116] (5/11/2015). Public Library of Science - About, [Online]. Available: https://www.plos.org/about/.

[117] (5/06/2015). LIVIVO - About, [Online]. Available: http://www.livivo.de/app/misc/help/about.

[118] (5/07/2015). AGRICOLA - About, [Online]. Available: http://agricola.nal.usda.gov/help/aboutagricola.html.

[119] (5/07/2015). International System for Agricultural Science and Technology - About, [Online]. Available: http://agris.fao.org/content/about.

[120] (5/07/2015). ZB MED - legal notice, [Online]. Available: http://www.zbmed.de/en/legal-notice/.

[121] (5/11/2015). BioMedCentral - About, [Online]. Available: http://www.biomedcentral.com/about.

[122] (5/11/2015). BioMedCentral - Data Mining, [Online]. Available: http://www.biomedcentral.com/about/datamining.

[123] (5/11/2015). PubGet - About, [Online]. Available: http://goto.copyright.com/AboutPubget.