h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Hochschule Darmstadt
- Fachbereich Informatik -

A Clinical Decision Support System for Personalised Medicine

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

vorgelegt von
Johannes Idelhauser (743870)

Referent:	Prof. Dr. Bernhard Humm
Korreferent:	Prof. Dr. Ronald Moore
Ausgabedatum:	11.04.2016
Abgabedatum:	11.10.2016

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.

Darmstadt, 11. Oktober 2016

Johannes Idelhauser

Abstract

Mit der stetig steigenden Zahl an medizinischen Ressourcen und Publikationen wächst die Herausforderung für Ärzte immer auf dem neusten Stand der Forschung und der aktuell empfohlenen Behandlungsmethoden zu bleiben. Zusätzlich ist in den letzten Jahren die Patientensicherheit ein stetig wachsendes Thema und einer der Ziele von personalisierter Medizin. Um Ärzte und andere klinische Bedienstete bei ihren Entscheidungen zu unterstützen, wird in dieser Thesis ein klinisches Informationssystem für personalisierte Medizin vorgestellt. Es versucht den Informationsbedarf von Ärzten durch die Integration und Aggregation von sekundären medizinischen Quellen basierend auf einer elektronischen Patientenakte (EHR) zu beantworten.

Das vorgestellte System besteht aus verschiedenen Services, die jeweils andere Informationsbedarfe abdecken. Einer dieser Services ist der Literatur-Service, der relevante und nützliche medizinische Publikationen für einen spezifischen Patienten findet. Andere Services sind unter anderem ein Medikationsinformations-Service, ein Arzneimittelinteraktion-Service und eine automatische Suchmaschine für relevant klinische Studien in der Nähe. Die Relevanz des vorgeschlagenen Systems wird durch die Implementierung zweier Services in einer EHR-Applikation für die Behandlung von Hautkrebs verdeutlicht. Die beiden implementierten Services sind der Literatur-Service und der Arzneimittelinteraktions-Service.

Besonderer Fokus wurde dabei auf die Usability des Informationssystems gelegt. Ärzte und andere klinische Helfer sollen schnell und intuitiv alle relevanten Informationen mit minimaler Systeminteraktion finden. Eine erste Studie mit medizinischen Anwendern deutet auf die Relevanz des Systems hin.

Abstract

Given the rapidly growing number of medical publications and resources, physicians face challenges in keeping up-to-date with current research and patient care best practices. Additionally, the topic of patient safety has been growing more and more important over the last years and is an important goal of personalised medicine. To support physicians and other health personnel in making clinical decisions, this thesis presents the concept and prototypical implementation of a Clinical Decision Support System (CDSS) for personalised medicine. It satisfies information needs of consultants at the point of care by integrating secondary medical resources based on a concrete patient's Electronic Health Record (EHR).

The proposed system consists of different services that satisfy different information needs. One of them is a literature service that provides relevant and useful medical papers that fit the patient at hand. Other services include but are not limited to a drug information service, a drug interaction checker and a clinical trial service that searches potentially beneficial trials in the country. The feasibility of the system is demonstrated by two example services that have been implemented in an EHR application for melanoma care. Those are an automatic medical literature search service and a drug interaction checker.

Particular focus has been on the usability of the CDSS allowing physicians and other health personnel to quickly and intuitively gather relevant information with minimal system interaction. An initial assessment of the CDSS by medical professionals indicates its benefit.

Parts of this thesis were published in the *Proceedings of the 2016 European Collaborative Research Conference (CERC 2016)*:

Idelhauser, J., Beez, U., Humm, B. G., & Walsh, P. (2016). "A Clinical Decision Support System for Personalized Medicine". In: U. Bleimann, B. Humm, R. Loew, I. Stengel, & P. Walsh (Eds.), *Proceedings of the 2016 European Collaborative Research Conference (CERC 2016)* (pp. 132–145). Cork, Ireland.

Contents

List of Figures

List of Tables

Acronyms

ARFF	Attribute-Relation File Format
BoW	Bag of Words
CDSS	Clinical Decision Support System
DCS	Document Clustering Server
DLL	Dynamic Link Library
EBM	Evidence-based Medicine
EF6	Entity Framework 6
EHR	Electronic Health Record
EMR	Electronic Medical Record
IDF	Inverse Document Frequency
IR	Information Retrieval
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LOC	Lines of Code
MCC	Matthews Correlation Coefficient
MDT	Multidisciplinary Team
MeSH	Medical Subject Headings
ML	Machine Learning
NB	Naïve Bayes
NLM	U.S. National Library of Medicine
NLP	Natural Language Processing
ORM	Object-relational Mapping
PMID	PubMed ID
POS	Part of Speech
REST	Representational State Transfer
SVM	Support Vector Machine

Chapter 1

Introduction

The topic of patient safety and care quality has been growing more and more important over the last years. One example for this is the just recently introduced German law that entitles patients with more than three prescribed drugs to have a medication chart drafted for them (BMG, 2016). However, not only in Germany the topic of patient safety is getting more important. Studies suggest that many medical errors could be avoided with the correct use of information technology in clinical settings (Lyman et al., 2010). This is mainly the introduction of Electronic Health Records (EHR) to store patient-specific data and the use of Clinical Decision Support Systems (CDSS) (Berner et al., 2007, p. 3) to support health personnel with the right information at the right time.

1.1 Motivation

In addition to the growing importance of patient safety, physicians face challenges in keeping up-to-date with current research and patient care best practices due to the rapidly growing number of medical publications and resources. Especially in the context of personalised medicine and cancer care, ongoing research leads to constant new insights and new treatment options. To ensure good treatment outcomes and prevent malpractice lawsuits (Marchant et al., 2013), physicians should therefore keep current with ongoing developments.

The domain of personalised medicine aims at tailoring medical decisions, practices, interventions or products to the individual patient based on their predicted response or risk of disease with the goal of increasing patient health and safety (Academy of Medical Science, 2015). While tailoring the treatment to individual patients is common practise in medicine, the term has been recently used for informatics approaches in medicine that use large amounts of patient data, particularly genomics data, for selecting appropriate therapies.

To support physicians and other health personnel in making clinical decisions, keeping current with ongoing research and new personalised medicine approaches, this thesis presents the concept and prototypical implementation of a CDSS for personalised medicine. It satisfies physicians' information needs at the point of care by aggregating and integrating primary (e.g. clinical trial results, original articles) and secondary medical resources (e.g. systematic reviews, drug monographs) based on a concrete patient's EHR, thus paving the way for personalised medicine. The proposed CDSS is integrated into an EHR application for melanoma skin cancer treatment.

1.2 Project Surrounding

The thesis is part of the project SAGE-CARE (*SemAntically integrating Genomics with Electronic health records for Cancer CARE*) which is funded by the European Commission, Horizon 2020 Marie Skłodowska-Curie Research and Innovation Staff Exchange, under grant no. 644186. As part of the project, an EHR application for treating melanoma cancer patients is currently developed (Humm et al., 2015; Beez et al., 2015).

In this context, previous work on providing medical publications for an EHR has been done in a master's thesis (Beez, 2015). That thesis discussed, amongst others, the architecture and implementation of a literature service in a commercial EHR application. Its author proposed a combined rule-based and machine learning approach for generating queries against the medical search engine *PubMed*. This thesis uses the already existing application code as a basis and integrates, modifies and extends the functionality into the EHR application for melanoma cancer treatment currently developed in the project. As a result, some basic ideas like the combin-

ation of a rule-based with a machine learning approach for query generation must therefore be attributed to the author of the preceding work. However, extensive modifications were made and new developments were added as part of this thesis.

1.3 Outline

The remainder of this thesis is structured as follows. Chapter 2 specifies the problem statement in terms of the requirements by describing functional as well as non-functional requirements. Chapter 3 outlines relevant background information and theoretical foundations. Physician's information demands and ways to satisfy them are analysed in Chapter 4. Subsequently, the proposed CDSS's architecture is outlined in Chapter 5 and finally implemented in Chapter 6. Chapter 7 evaluates the proposes CDSS and its implemented services by comparing it against the requirements using feedback from medical experts while Chapter 8 compares the thesis' approach with related work. Finally, Chapter 9 concludes the thesis and indicates future work.

Chapter 2

Requirements

As described in Chapter 1, the goal of this thesis is to provide a CDSS for supporting physicians and other health personnel during patient encounters. In discussions with clinicians involved in the treatment of melanoma as well as medical students, the following requirements for a CDSS integrated into an EHR application were identified:

1. **Functional Requirements**

    1.1. *Relevant*: The CDSS shall satisfy the consultants' information demand with relevant, helpful and latest medical information.

    1.2. *Personalised*: The information provided shall be tailored to the medical condition of a particular patient.

    1.3. *Pro-active*: The CDSS shall offer information pro-actively without additional data entry by the user.

    1.4. *Easily comprehensible*: shall provide a quick overview of all information available as well as the possibility to easily acquire more detailed information where needed.

    1.5. *Workflow*: The CDSS shall not interfere with the consultant's EHR workflow.

2. **Non-Functional Requirements**

    2.1. *Usable*: The CDSS shall be intuitive to use and self-explanatory.

2.2. *Low response time*: The response time for all interactions with the CDSS shall be less than 1s.

2.3. *Extensible*: The ongoing extension of the CDSS with new information sources shall be facilitated with moderate implementation effort.

Chapter 3

Background

This chapter introduces important definitions and background concepts of this thesis. First, the terms Personalised Medicine (Section 3.1), Electronic Health Records (Section 3.2) and Clinical Decision Support System (Section 3.3) are explained and discussed. Then, the foundations of relevant machine learning techniques are investigated (Section 3.4). As this thesis also processes natural language text, Section 3.5 introduces useful Natural Language Processing (NLP) techniques. Finally, Section 3.6 describes relevant concepts of Information Retrieval.

3.1 Personalised Medicine

The term *personalised medicine* is often used but defined in many different ways. Many definitions associate the term with genetics and gene sequencing to predict a patient's response to a drug or therapy. Especially developments in the field of genetics lead to significant drops in gene sequencing costs. This allows the development of tailored treatments and drugs to target special gene mutations like *BRAF* or *ALK* and significantly improves drug response and survival rates (PMC, 2014).

Other definitions view the term *personalised medicine* as something that always existed because medicine has always seen the individual patient in its centre. Redekop et al. (2013) developed a more general definition and define personalised medicine as "the use of combined knowledge (genetic or otherwise) about a person to predict disease susceptibility, disease prognosis, or treatment response and thereby improve

that person's health." (Redekop et al., 2013). It is therefore not solely genetic information that is used to predict the response in personalised medicine, but also other factors like the patient's proteins, environment or other indicators not yet known. Following their logic, truly personalised medicine is the ultimate goal and ideal form of medicine that can only be achieved with further advances in science.

3.2 Electronic Health Records

As this thesis implements and integrates a CDSS into an existing EHR application, the term ERM should first be properly defined. The ISO (International Organization for Standardization) defines an EHR as a "repository of information regarding the health status of a subject of care, in computer processable form, stored and transmitted securely and accessible by multiple authorized users" (ISO, 2005). It continues to mention the use of a standardised EHR-independent information model to support health care integration efforts.

However, others see the EHR more as a tool spanning multiple independent organisations that holds all long-term records of a patient with the aim to facility care across institutions (Rishel et al., 2005). Literature suggests there is a difference between an electronic *health* record and an electronic *medical* record (EMR). An EHR is considered as focusing on the overall health of the patient and provide a broader view on the patient's care by sharing information across organisations. EMRs on the other hand are more focused on treatment in practice. They are basically electronic versions of the paper charts in clinician's offices but usually do not leave the clinic in electronic form (Garret et al., 2011).

In the context of this thesis the ISO definition is considered appropriate. This work will refer to an EHR application to describe the whole software storing data for several patients and mention an EHR when the actual patient's data is meant.

3.3 Clinical Decision Support Systems

A CDSS provides "clinicians, staff, patients, and other individuals with knowledge and person-specific information, intelligently filtered and presented at appropriate

times, to enhance health and health care" (Berner, 2009). An important aspect of the CDSS definition is the fact that a CDSS does not aim at replacing professional advice but rather seeks to support health personnel in making decision and therefore helps reduce medical error (Karthigeyan et al., 2014). The target group of a CDSS are not only physician's but also other health personnel, e.g. nurses. In fact, it was found that in some cases the use of a CDSS was more effective if it also provided nurses with information (Berner, 2009, p. 6).

Considering classical decision support systems, several different types can be identified based on the underlying data source (Power, 2008; Sanchez, 2014):

- **Model-driven systems**: They utilise mathematical, probabilistic or machine learning models that abstract a problem and allow the user to manipulate model parameters in order to see the outcome and make decision.
- **Data-driven systems**: Using huge amounts of data and data mining or business intelligence techniques to provide the management with insight into their organisation.
- **Communications-driven systems**: Their focus is on collaboration, communication and social interaction to reach a decision. This is also important for clinical decision support systems as physicians tend to have confidence in the knowledge of trusted colleagues.
- **Document-driven systems**: Those systems leverage document retrieval and analysis techniques to navigate over document databases like research papers, standards, guidelines, etc.
- **Knowledge-based systems**: Leverage a formalised knowledge base in a special representation, these systems usually use ontologies to infer and and reason over.

This classification can also be applied to clinical decision support systems although the distinction made most often is between knowledge-based and non-knowledge-based systems (Berner, 2009, p. 5).

On top of that, a variety of systems and modules can be regarded as clinical decision support. Current implementations include drug monographs, drug dosage calculation based on the patient's age and weight or alerts and reminders systems with

manually added rules. CDSS development and architecture in recent years tended to shift from stand-alone solutions to ones integrated in clinical systems that leverage standards in order to reuse existing CDSS modules. The most recent development shows efforts to build CDSS systems in a service-based architecture (Wright et al., 2008).

3.4 Machine Learning

The task of Machine Learning (ML) is a branch of Artificial Intelligence and finds patterns or makes predictions for the future from data. This section gives a short introduction to the domain of ML. From several main ML areas, two are relevant in the context of this thesis: supervised and unsupervised learning. Of those, supervised classification (Section 3.4.1) and unsupervised clustering (Section 3.4.2) are used in this work.

3.4.1 Supervised Classification

In supervised learning the task is to learn a function that can assigns outputs (also called labels) to given inputs (also called feature vectors) using training data that has both input and output variables. The learned function can later be used to assign labels to unseen data. Typical examples of supervised learning are classification and regression tasks where the output variables are either labels (classification) or continuous valued numbers (regression) (Sammut et al., 2011, p. 941). The following section describes the case of classification.

Training a Classifier

Usually, training a classifier utilises pre-labeled data which is split into a training and a test data set. As the name suggests, the first is used to train the ML model while the test data set is used to evaluate the quality of the generated model on unseen data. If enough labeled data is available, this separating is done by splitting the data set at a certain percentage into training and test data. However, if there is not enough labeled data, the process of cross-validation can be chosen. Cross-validation separates the whole labeled data set into n partitions, called the *folds*.

Then the classifier is trained and evaluated n times by always taking $n-1$ folds as training an the remaining fold as test data. The classifier is then evaluated by combining all n results.

Multi-Class and Multi-Label Classification

Many classification tasks are binary and only assign one from two possible labels to each unlabelled input instance. Examples for that are spam or fraud detection classifiers. However, there are also many real world problems that are not binary classification problems. The number of possible labels is usually greater than two. If from the many labels only one is assigned to each instance, the task is generally referred to as *multi-class classification*. Whereas if more labels are assigned to an instance, the task is usually a *multi-label classification*.

3.4.2 Unsupervised Clustering

Unsupervised learning describes the more explorative approach of finding patterns in unseen data without having outputs like labels in supervised learning. Common tasks of unsupervised learning include clustering and dimensionality reduction (Sammut et al., 2011, p. 1009). Clustering is often used for knowledge discovery in data exploration. It tries to partition an unseen dataset into groups of similar instances, called clusters. Large data sets like for example social networks are often clustered to identify communities of people (Mohri et al., 2012, p. 2). Similarity between instances is typically calculated by applying distance measures like the Euclidean or Manhattan distance (Sammut et al., 2011, p. 180).

3.4.3 Algorithms

There are many different ML algorithms for different ML tasks and each of them has its own strengths and weaknesses. In the context of this thesis, two algorithms are used that are explained further in this section.

Naïve Bayes

Naïve Bayes (NB) is a simple classification algorithm that is based on the Bayes theorem and assumes a strong independence among features, given a class. Although this independence is often not present in real world data sets, it often delivers competitive results and is widely used in practice. Computational efficiency and the robustness in the face of noisy and missing data help to make it more attractive in practice (Webb, 2010, p. 713).

Support Vector Machines

Support Vector Machines (SVM) are typically linear algorithms that are used for classification and regression tasks. In the simplest classification task with only two classes, SVMs try to find a hyperplane with the biggest margin that separates the two classes. As seen in Figure 3.1, both hyperplanes *H1* and *H2* correctly separate the two classes. However, *H2* has a wider margin than *H1* and will probably generalise better on unseen data (Zhang, 2010, pp. 941)

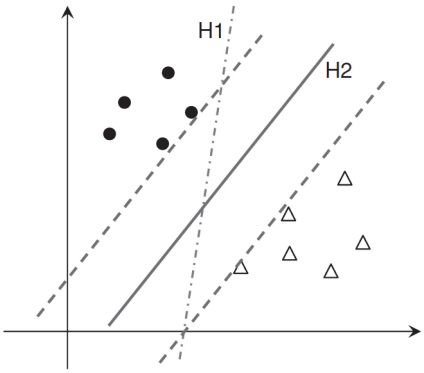

Figure 3.1: Support Vector Machine (Zhang, 2010, p. 943)

Especially in text categorisation SVMs prove their superiority and often "deliver state-of-the-art performance in accuracy, flexibility, robustness, and efficiency" (Zhang, 2010, p. 942). As real world problems are often not linearly separable, SVMs can be extended by using appropriate kernel functions. Additionally, SVMs have been extended to cope with multi-class and multi-label classification problems.

3.4.4 Performance Metrics

Supervised learning methods are usually evaluated by calculating different measures based on the confusion matrix (Table 3.1). A confusion matrix has a size of $n \times n$ where n is the number of classes. Its rows represent the actual class labels of each instance whereas the columns show the predicted classes. The content of a confusion matrix consist of TP (true positive), the number of correctly identified positive instances and TN (true negative), the number of correctly labeled negatives. Similarly, FN (false negative) and FP (false positive) describe falsely identified instances (Rechenthin, 2014, p. 58).

		Predicted classes	
		Yes	No
Actual classes	Yes	TP	FN
	No	FP	TN

Table 3.1: Confusion matrix

Popular measures for determining the quality of a classifier are *precision* and *recall*. They are not only used in ML, but also in the domain of IR. Precision describes the percentage of instances that are correctly classified as positives while recall describes the percentage of correctly classified positives out of all positives. High recall typically occurs with low precision and high precision with low recall. To accommodate this, the *F-Measure* builds the harmonic mean of the two. A value of 1 indicates a classifier having perfect scores for both precision and recall (Rechenthin, 2014, pp. 60):

$$\begin{aligned} Precision &= \frac{TP}{TP + FP} \\ Recall &= \frac{TP}{TP + FN} \\ F\text{-}Measure &= \frac{2 \times precision \times recall}{precision + recall} \end{aligned} \tag{3.1}$$

The last measure, *Matthews Correlation Coefficient* (MCC) was introduced by Matthews (1975) to measure the quality of a binary classification, especially in cases where the classes are of hugely differing sizes. It shows the correlation coefficient between predicted and actual binary classes and returns values between -1 and +1.

+1 relates to perfect agreement between the variables, -1 to total disagreement. It returns 0 for completely random predictions (Baldi et al., 2000, p. 415).

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{3.2}$$

3.5 Natural Language Processing

Parts of this thesis, like text classification, document clustering and IR, use or mention tools and techniques stemming from NLP. This section introduces those used in this work. It is worth noting, that most of the presented techniques are dependent on the text's language and have to be considered differently for each individual language.

Tokenisation

Tokenisation splits a text into individual pieces, called tokens. In many languages, including English, using the space character is a good approximation to split the text. However, there are special cases as for example in the sentence "Hello, this is a test.". Simply splitting at the spaces results in two tokens including some punctuation: "Hello," and "test.". Another example that tokenisation is not a trivial task is splitting "aren't" into tokens. The possibilities include "aren't", "arent", "are n't" and "aren t". The question is which would be best to represent the original text and needs special consideration, for example when building a search engine (Manning et al., 2008, pp. 22).

Sentence Segmentation

Sometimes a text has to be separated into individual sentences. This is usually referred to as *sentence segmentation* or *sentence splitting*. Although in languages like English full stops and other punctuation characters like "?", ":" or "!" can be used, the task is not trivial. For example, simply splitting "Mr. & Mrs. Smith like to go to the cinema." at the period character will result in a wrong segmentation.

Therefore, a ML model is often trained that can be used to predict if a specific sign indicates the end of a sentence or not.

Stemming and Lemmatisation

Usually, natural language text consists of words in different grammatical versions, e.g. *organise*, *organises* or *organising*. Similarly, different words might hold a similar meaning as for example *democratic*, *democracy* and *democratisation*. In some situations it might be useful to not store each variant of those word but rather only one word representing all of the different versions. This would lead to a dimensionality reduction of a document's BoW representation. For this, stemming and lemmatisation are used to transform different related forms of a word into a common base form. Where stemming does this by simple pruning of the words, lemmatisation uses dictionaries and morphological analysis of words (Manning et al., 2008, p. 32).

POS Tagging

A *Part-of-speech* (POS) tagger assigns the part of speech to a word in a text. As POS labels depend on the word's definition but also its context in the sentence, a POS tagger usually utilises a trained model to assign the most likely POS tag (Charniak, 1997). POS tags are usually short symbols like "VB" for a verb in the base form or "NNS" for a plural noun[1]. POS tagging is the basis for more advanced natural language processing like syntactical sentence parsing. This is however not relevant in this context.

Bag Of Word Models

Documents or text can be represented in a BoW model when only the occurrence of terms but not their position in the text is important. As a result, the sentence "Mary is quicker than John" results in the same BoW model as "John is quicker than Mary". This is acceptable as documents with a similar BoW model tend to be similar in content (Manning et al., 2008, p. 117). As an example of how a BoW

[1] A detailed list of POS tags often used can be found under `https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`

model works, the sentence “Mary likes to be quicker than John” is added to the sentence collection and the following list of words is created from them:

```
["Mary", "is", "quicker", "than", "John", "likes", "to",
    ↪ "be"]
```

Using the list above, the following lists can be created for each sentence:

```
//Mary is quicker than John
[1, 1, 1, 1, 1, 0, 0, 0]
//John is quicker than Mary
[1, 1, 1, 1, 1, 0, 0, 0]
//Mary likes to be quicker than John
[1, 0, 1, 1, 1, 1, 1, 1]
```

3.6 Information Retrieval

Some aspects of this thesis are located in the domain of *information retrieval* (IR). The goal of IR is usually to find information (usually in the form of documents) in a unstructured form (usually natural language text) from a large collection or database. The retrieved resources should hereby help the user answer his or her information need and considered relevant if they do (Manning et al., 2008, p. 1).

This section describes the parts of IR relevant for this thesis. First, information retrieval models by which IR systems can be categorised are illustrated in Section 3.6.1. Section 3.6.2 presents evaluation metrics that can be used to measure the performance of IR system.

3.6.1 Retrieval Models

Three main retrieval models can be identified in IR. The Boolean model, the vector space model and the probabilistic model. The first two models are relevant for this thesis and therefore described in more detail.

Boolean Model

The Boolean retrieval model is based on set theory and Boolean algebra used in the past for many early retrieval systems. The idea behind the Boolean retrieval model is to view a document as a set of index terms that are either present or absent. The index term weights are therefore all binary. Queries are basically Boolean expressions consisting of index terms combined with the connectors `AND`, `OR` and `NOT`. Evaluating these expressions results in a binary decision where each document is either relevant or non-relevant to a query without any partial matches. Staying in the domain of Boolean algebra, a query Q can be expressed as a Boolean expression in disjunctive normal form. The query $Q = t_a \wedge (\neg t_b \vee t_c)$ could therefore be expressed as $(1,0,0) \vee (1,0,1) \vee (1,1,1)$, with each term being a binary vector of the form (t_a, t_b, t_c). This can be seen in Figure 3.2 (Sugiyama, 2004).

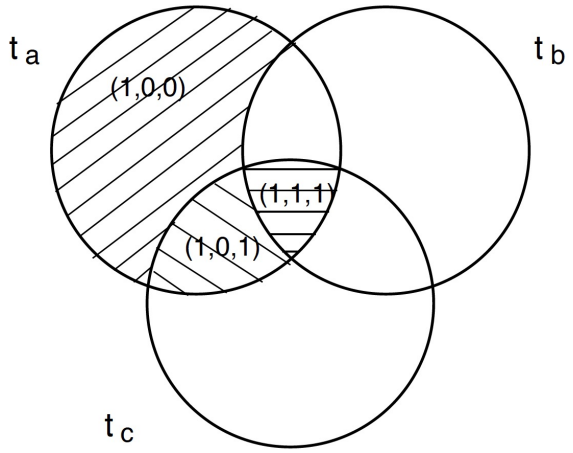

Figure 3.2: Boolean set theory for the query $Q = t_a \wedge (\neg t_b \vee t_c)$ (Sugiyama, 2004)

Drawbacks of the Boolean model include the binary decision model where documents are considered either relevant or non-relevant to a query without the notion of ranking. This hinders good retrieval performance as in IR there is often not this clear distinction. For example, documents might not be found if they do not contain an index term specified in the query. The document could nevertheless be important as it can contain a synonym of the query term while also presenting all other query terms. This exact matching and subset creation might lead to too many or too few documents being returned. Additionally, users often consider it difficult to formulate their information need in a Boolean query. As a result, the generated user queries are usually rather simple. Advantages of the model include a clean formalism and

comparably simple implementation (Sugiyama, 2004, p. 12). Current IR systems know of the drawback of the Boolean retrieval model and try to find different ways of index term weighting.

Vector Space Model

In contrast to the Boolean retrieval model, the vector space model recognises the limitations of binary term weighting and assigns non-binary weights to the index terms. The assigned term weights are used to calculate the similarity between the query and the documents in a retrieval system. This similarity measure is used to rank the returned documents and therefore account for documents only partially matching the query. It is also a good way of providing order to a large retrieved collection by showing the documents more similar to the query first.

Term Frequency The simplest approach to assign weights to index terms is to count the number of occurrences of a term t in a document d. This is called the term frequency $TF_{t,d}$. Counting the number of term occurrences without any preservation of the order they are in, is usually called creating a *bag of words model* (BoW). However, not all terms in a document are equally important in determining the relevance of it. One example are *stop words* that are excluded from indexing as they are too common words and carry little meaning for indexing a document. Exemplary stop words are prepositions like “of”, “a”, “on”, ... (Manning et al., 2008, p. 117).

Inverse Document Frequency Simply considering the mentioned term frequencies to asses the relevancy on a query has one drawback as all terms are considered equally important. This is in fact not the case as for example the term “auto” occurs in almost all documents of a document collection about the automotive industry. To accommodate this, the inverse document frequency (IDF) value idf_t of a term t is calculated over the whole document set by looking at the number of all documents N and the number of documents containing the term df_t:

$$idf_t = log \frac{N}{df_t}$$

TF-IDF Weighting Using the two previously mentioned measures tf and idf, a combined weight for each term in each document is calculated by

$$\underset{t,d}{tf\text{-}idf} = tf_{t,d} \times idf_t$$

Applying $tf\text{-}idf_{t,d}$ to a term t in a document d results in the following values:

- highest value, if the term occurs many times in a small set of documents
- lower value if the term occurs a few times in many documents
- lowest value if the term occurs in practically all documents

The documents as well as the query can from now on be seen as vectors of $tf\text{-}idf$ values. This representation is considered to be the *vector space model* and is the basis for other IR tasks like scoring, document classification and document clustering. Considering $\vec{v}(d)$ as the vector generated from document d, the documents and the query can be displayed in a vector space with one dimension per term (Figure 3.3). To calculate the similarity between vectors and account for different document lengths, the standard way is to use the *cosine similarity* of two vectors (Manning et al., 2008, pp. 119).

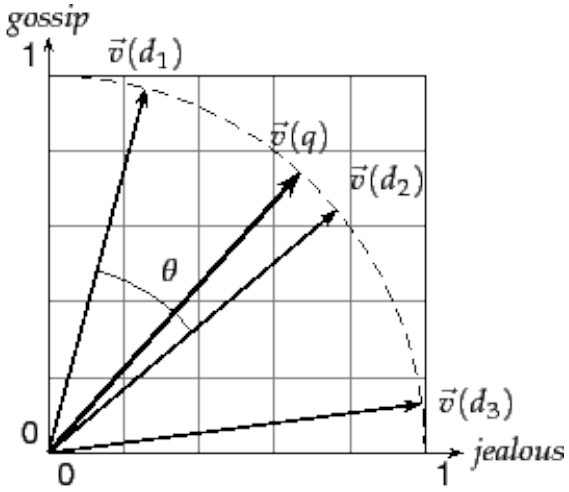

Figure 3.3: Illustrated cosine similarity $sim(d_1, d_2) = cos\theta$ (Manning et al., 2008, p. 121)

3.6.2 Evaluation Metrics

Although in IR the user query is usually rather vague and the returned documents are not exact answers to a specific question, the retrieval system has to be evaluated

by looking at the relevance of the retrieved documents regarding the query. The most popular measures in accessing the quality of a IR system are precision, recall as well as the harmonic mean of the two. Measuring is usually done by defining a test collection and test queries an letting experts decide what documents are relevant for the test queries.

Precision

The precision is the fraction of retrieved documents that are relevant, answering the question of how many of the retrieved documents are relevant:

$$Precision = \frac{\#(relevant\ items\ retrieved)}{\#(retrieved\ items)}$$

Recall

The recall measure resembles the fraction of relevant documents that are retrieved, effectively stating how many relevant items were selected.

$$Recall = \frac{\#(relevant\ items\ retrieved)}{\#(relevant\ items)}$$

Harmonic Mean (F-Measure)

As focussing on any of the previously mentioned evaluation metrics might be too focused, the *F-measure* calculates the harmonic mean between the two:

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

The three measures only consider unranked retrieval where the whole unordered set is used to calculate the measures. However, in a more realistic IR system the user is typically only interested in the the top k returned documents, hence the ranking plays an important role in assessing the quality of the system. As a user usually proceeds from the top to the bottom of the retrieved ranked document list, the precision and recall will vary depending on the current position and the number

of relevant or non-relevant documents so far seen. To evaluate ranked systems, the precision and recall values are gathered for different evaluation steps, usually based on the 11 standard recall levels 0%, 10%, 20%, ..., 100% and plotted to form a *precision-recall curve* (Figure 3.4)(Sugiyama, 2004, pp. 23).

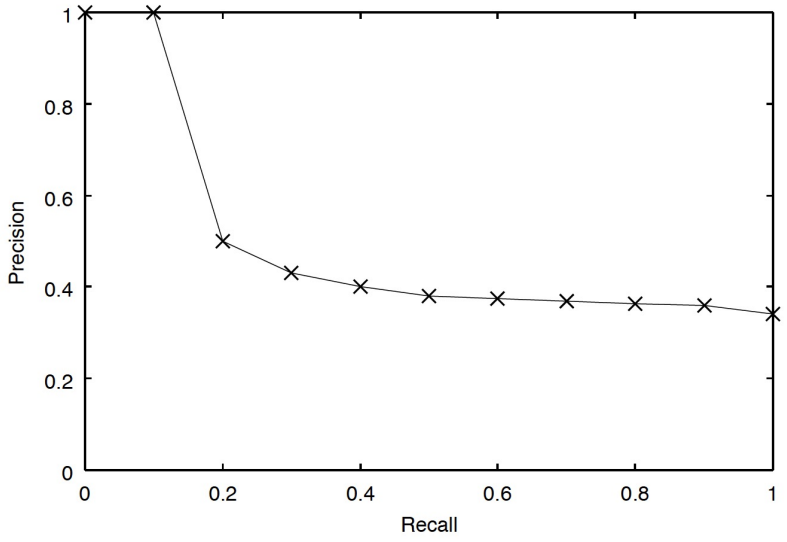

Figure 3.4: Precision-recall curve at 11 standard recall levels (Sugiyama, 2004, p. 25)

Chapter 4

Information Architecture

This chapter describing the information architecture first investigates common clinical information needs and physician's questions at the point of care (Section 4.1). Section 4.2 then presents an interaction concept that tries to satisfy these found information demands at the point of care. Potential available information sources are identified and mapped on the information demands in Section 4.3.

4.1 Information Demand

A central question when building a CDSS is which information medical professionals might need at the point of care. Several studies try to answer that question in order to improve medical education. Clarke et al. (2013) found that most physicians' and nurses' information demands could be related to the domains of diagnosis, medication and therapy. Similarly, Ebell et al. (2011) reported that "[t]he most common question types were 'How should I treat finding/condition y given situation z?', 'Is drug x indicated in situation y or for condition y?', and 'What is the cause of symptom x?' " (Ebell et al., 2011). Information sources should therefore "provide relevant, valid material that can be accessed quickly and with minimal effort" (Smith, 1996).

Maggio et al. (2014) interviewed 21 physicians from the US and the Netherlands and identified six information needs at the point of care (Table 4.1). Their secondary objective was to determine which kind of data source is rather used to satisfy these information needs. They focus on *PubMed* as the source for primary literature and

UpToDate as the source for evidence-based summaries. It is notable that both data sources seem to have a right to exist and are used frequently to answer different kinds of questions. The authors also mentioned that often those resources were not checked during patient encounters due to time constraints, but were rather consulted after work.

Reason	Definition
1. Refreshing	To update or aid in the recall of one's own known knowledge
2. Confirming	To check one's own knowledge for self-satisfaction or in preparation to speak, take action, advise patients, etc. To confirm another individual's or resource's knowledge/coverage of a topic
3. Logistics	To answer practical questions to facilitate action
4. Teaching	To teach trainees through a variety of methods, including lecturing, role modelling, etc
5. Idea generating	To generate ideas for treatment, diagnosis or an overall sense of what is happening with a patient
6. Personal learning	To foster one's own learning or satisfy curiosity

Table 4.1: The six information needs identified according to Maggio et al. (2014)

Another study focused on drug information needs at the point-of-care in Sweden (Rahmner et al., 2012). Identified as important were automatically generated alerts for severe drug interactions or adverse effects. The alerts should be linked to the EHR as physicians would otherwise not take the time to check for interactions. Also, the severity of the interaction would be an important factor when displaying alerts as too many non-relevant alerts would lead to an alert fatigue. As a result, physicians might ignore even severe interactions. Another aspect they noted was the question of how the drug will affect the patient. Hereby, physicians wished to see information about the reversibility of severe adverse effects. To detect if a patient's symptom is drug related or not, the possibility to search a patient's drug lists for adverse effects matching the symptom was proposed (Rahmner et al., 2012).

Concerning drug safety, physicians wished data on allergy or hypersensitivity. This was actually perceived as most important as the system should actively stop prescription when such a case would occur. In order for that to work, data on the patient's allergies would have to be stored in the EHR and filled when an allergy is identified. Drug dosing is especially difficult in children and elderly patients as there is a narrow drug range and limited research in that area. Physicians' wished a system to link between weight and drug dose to calculate the dosing and create

automatic alerts. Other functionalities included drug images to provide patients with information on how good a pill can be swallowed and if it could be split (Rahmner et al., 2012). Asked for ways to improve existing EHRs, participants of the study suggested "links and/or sheets to scientific articles and news about certain drugs" (Rahmner et al., 2012, p. 122).

Due to the rapid progress and development of new drugs and therapies in cancer treatment, patients and physicians are encouraged to participate in clinical trials (National Cancer Institute, 2016). Although this is not a specifically mentioned information need, it makes sense to include a service that searches relevant clinical trials for the patient based on its EHR.

4.2 User Interaction Model

To satisfy the previously identified information needs, physicians tend to access primary literature in the form of abstracts and full-text as well as secondary literature in the form of summaries and reviews (Maggio et al., 2013). In order to provide an intuitive way for physicians and other health personnel to access this information, the proposed CDSS leverages several information services that each try to satisfy different information needs. Those information services are organised in panels that the users can customise and fit to their needs by deciding which service panels should be displayed and which should be hidden. Additionally the order and size of the panels can be adapted to the user's individual needs while the resulting layout is persisted over different sessions for the individual user.

4.2.1 Literature Service

One of the main services in the CDSS is the *literature service* to find and display relevant primary medical literature that is related to the patient at hand (Fig. 4.1). The literature service displays automatically generated filters to quickly navigate the found literature. The filters are displayed on the left whereas the medical literature is shown on the right. For each medical publication its title, journal and publication date is displayed. In the context of evidence-based medicine (EBM), publications with a high degree of evidence are to be preferred in patient care (Hung et al.,

2015). As such, publications that are reviews or clinical trials are shown with a marker indicating their publication type. This also aligns with a study from 2013 that logged and analysed data queries in a hospital and found that "[a]lmost a third of the articles [...] accessed were reviews." (Maggio et al., 2013).

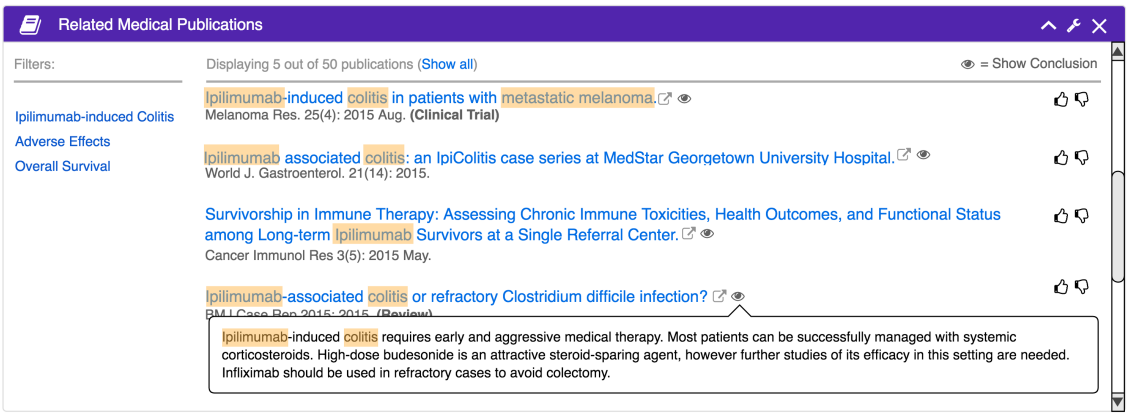

Figure 4.1: Literature service mockup

For quick orientation and relevance assessment, terms that appear in the patient's EHR are highlighted in the literature service. To help the relevance assessment process, a teaser text is displayed when hovering the eye icon after each publication title. In order to give the users a way to give feedback on the relevance of a publication and improve the literature search, icons with a thumbs-up and a thumbs-down are provided.

4.2.2 Evidence-Based Medical Recommendations

Evidence-based medicine describes the assessment and use of the best available research for decision making in patient treatment and diagnosis. This is done by focusing on well-designed, conducted research with a strong level of evidence like systematic reviews or randomised controlled trials (Hung et al., 2015). There a clinical decision support systems that specialise on providing evidence-based treatment guidelines and summaries written by medical experts that reflect the current state of research. Obst et al. (2013) assessed the use of such a service named *UpToDate.com* and suggested that physicians often have little time and that navigating the service could sometimes be time consuming. They also noted that the summaries were sometimes written confusingly and that it took too long to quickly grasp the desired information.

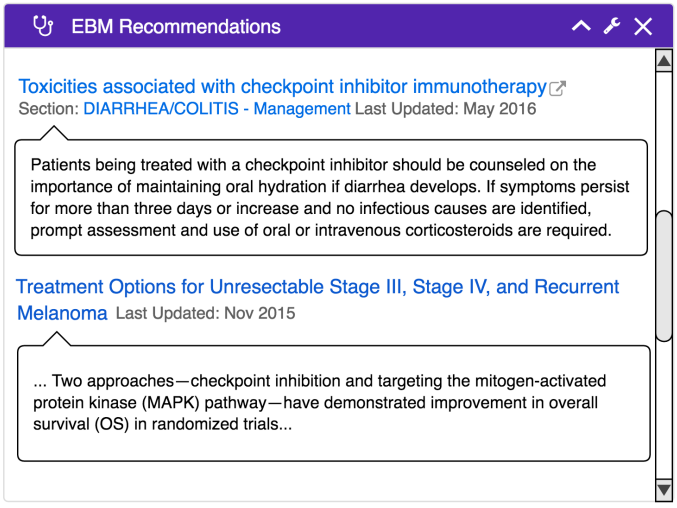

Figure 4.2: EBM Recommendations Service

The *EBM recommendation service* (Fig. 4.2) therefore queries different secondary information sources for patient-related evidence-based summaries and reviews and extracts the most important sections that describe the patient's issue. If the users wish to see the extracted sections in context, they can follow the links and visit the original text.

4.2.3 Drug Information Service

Other important information in patient care is information on drugs and their interactions "at the point of drug prescribing" (Rahmner et al., 2012). Therefore, the *drug information service* provides information normally available in package inserts and secondary decision support services in a more accessible and structured way (Fig. 4.3, left). The provided information includes dosage data for different age groups and pre-filled calculators to compute the correct dosage based on the age and weight of the patient. Other information consists of warnings, adverse effects, pregnancy, pharmacology, administration guidelines, material for patient education and pill images and prices. Selecting a drug for displaying can be done in an autosuggest-supported field that ranks already prescribed medication higher, but allows also searching for medication not yet prescribed.

As physicians indicated they wanted to see automatically generated alerts for severe drug interactions and adverse effects (Rahmner et al., 2012), an alert is displayed

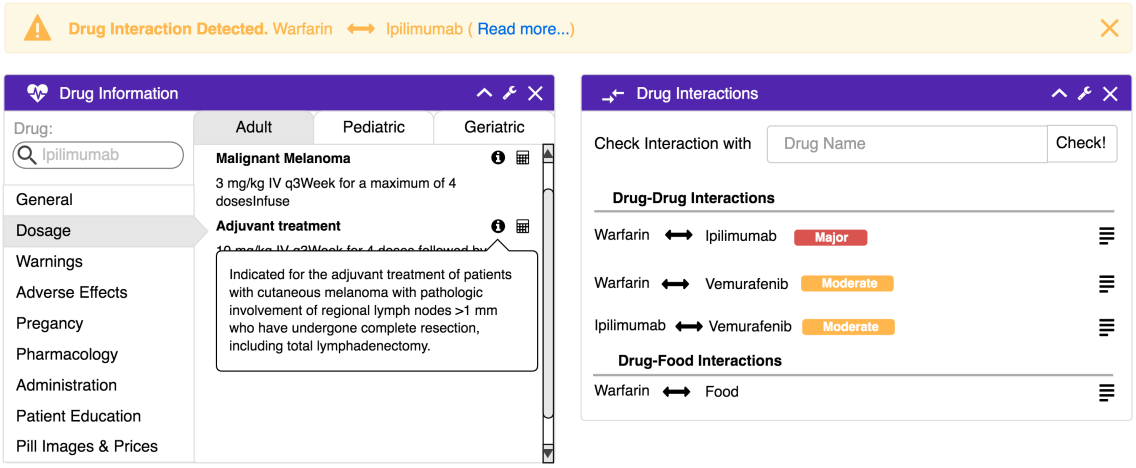

Figure 4.3: Drug information service with different service panels

prominently (Fig. 4.3, top). For more information on how to manage the interaction or alternative drugs, an appropriate link is provided. Non-severe drug interactions as well as drug-food interactions are displayed in an own panel where the users have the possibility to check interaction with other, not yet prescribed drugs (Fig. 4.3, right).

To "make drug information more searchable" (Rahmner et al., 2012) and for example allow checking if a patient's symptom could be drug related, an adverse effects panel is introduced (Fig. 4.4). It automatically identifies drug-related comorbidities that are registered in the EHR but also allows searching for symptoms not yet recorded. An option to read more provides information on how to manage this effect, when it will occur or how long it will last.

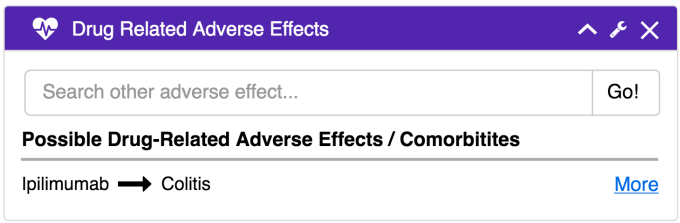

Figure 4.4: Searchable Adverse Effects Service

4.2.4 Clinical Trials Locator Service

Especially in cancer care the participation in clinical trials is an option for patients of all clinical stages as new findings lead to the development of many new drugs (National Cancer Institute, 2016). A *clinical trials service* is therefore introduced that searches for nearby clinical trials fitting the patient (Fig. 4.5, left).

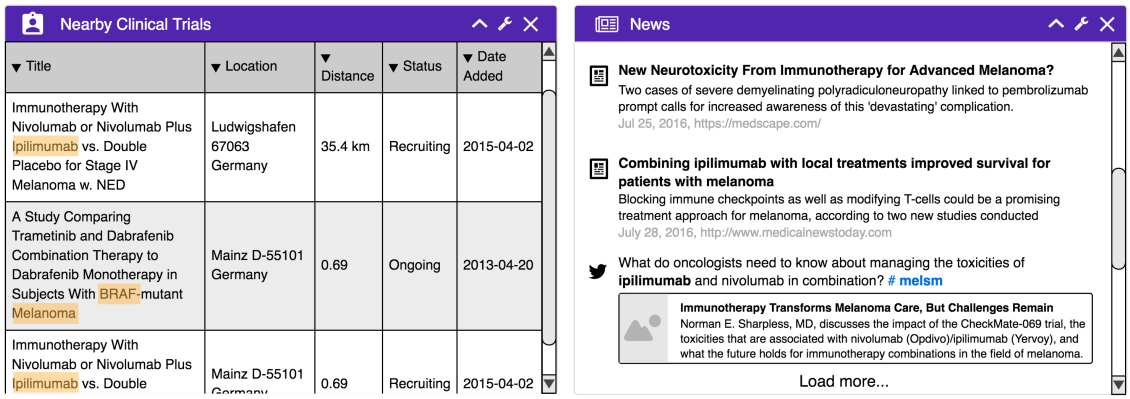

Figure 4.5: Clinical trials locator (left) and medical news service (right)

Found clinical trials are displayed by their title as well as the location and distance to the user's current location. Similar to previously mentioned services, terms appearing in the EHR are marked to allow quick visual navigation. Also, the status and the date the trial was added to the clinical trial register might be important to know information. By clicking on the title of a clinical trial the user can open the original entry to further assess its relevance.

4.2.5 Medical News Service

Finally, a news service provides recent news on treatments, drugs, legislative information or other scientific breakthroughs that can be related to the current EHR (Fig. 4.5, right). The service module displays the title and a little teaser text to allow quick relevance assessment as well as an icon describing the type of the information. To read the news, the user is directed to the services original site.

4.3 Information Sources

In order to provide the previously mentioned CDSS services, several potential information sources were identified. The selection of information sources is a critical part in any CDSS application as they are the basis for the trust given them by physicians. As patients' health might also be depended on this information, the quality and correctness of the information is obligatory. This section introduces a summary of identified relevant sources for a CDSS and potential pitfalls when using them.

4.3.1 Literature Service Sources

Asked about what information sources they use, health personnel reported a wide range of sources "with a strong preference for PubMed and UpToDate, as well as Google" (Maggio et al., 2013, p. 206). *PubMed* is a free search engine for biomedical databases with over 23 million publications and is perceived as the source for the most up-to-date research by physicians. However, it is also considered as challenging to use, especially in contrast to evidence-based summary services like *UpToDate.com*, wich will be discussed in Section 4.3.2 (Maggio et al., 2014). Nonetheless, *PubMed* is well-known in the medical community and frequently used. This could also be observed during the interviews with medical students and resident physicians.

Apart from *PubMed*, there are many other databases and search engines that focus on providing access to scientific medical publications. Differences between them include the type and number of queried databases, the included journals, their access type and the availability of an application programming interface (API). Appendix A gives an overview of identified primary literature search engines and databases that could be used in the scope of this thesis. It is however important to note that collections of different services are not distinct. For example, *PubMed* also includes articles from the *Cochrane Database of Systematic Reviews*.

4.3.2 Evidence-based Medical Recommendations

Evidence-based decision support services are often revert to as POC tools and combine primary literature evidence into summaries, reviews and clinical guidelines. They are designed to deliver "pre-digested, rapidly accessible, comprehensive, and

periodically updated information to health care providers" (Kwag et al., 2016). As evidence-based reviews or summaries should be written by medical experts to ensure quality and reliability, most services require a commercial licence or paid subscription to access them. Only a few public and free sources could be found in the scope of this thesis (Appendix B).

Kwag et al. (2016) assessed the quality and volume of 26 web-based POC information summarisation services and concluded that *BMJ Best Practice*, *Dynamed* and *UpToDate* showed the highest overall scores across all dimensions. Other examined services that were advertised as evidence-based were less reliable which is why they recommend to regularly asses the value and quality of used POC summary services. Nearly a quarter of the 26 investigated services were newly identified in 2014 which is a sign that the market is growing strongly. In an earlier study, Banzi et al. (2011) analysed the updating speed of the five most highly ranked point-of-care information summary services and found that *DynaMedPlus* was the most up-to-date service. Yet, they point out that it is not clear if high updating speed is also correlated to a better integration of this new information.

Selecting a data source for the EBM service might therefore not only depend on the ability to include it over an API and its cost, but also on the quality and subsequently the trustworthiness of the service.

4.3.3 Drug Information Sources

As drug information at the POC is an important topic, there are many services providing drug information in the form of drug monographs, drug interactions checker and other material. An overview of identified drug information sources can be found in Appendix C. As stated in Rahmner et al. (2012), incomplete drug coverage or missing drugs negatively affect the confidence in the reliability of the system. Therefore, the data sources would have to be carefully selected to ensure the highest possible data quality.

Peters et al. (2015) investigated the quality of two public drug interaction sources by comparing the drug interaction services *DrugBank* and *NDF-RT*. They found a limited overlap between the two services' reported drug interactions. The commercial service used to compare against provided better coverage of the test interaction

set than both of the free services combined. Additionally, as of September 2016, the second public data source *NDF-RT* removed the drug interaction information from their service. Furthermore, DrugBank does not provide information on the drug interactions' severity. Taking these points into account one could assume that commercial drug information services provide a better data quality.

Wang et al. (2010) investigated commercial drug interaction services and discovered discrepancies among major services in identifying severe drug contraindications mentioned in the drugs' black box warnings [1]. They concluded that "[c]linicians should consult multiple drug resources to maximize the potential for detecting a potentially severe drug interaction" (Wang et al., 2010, p. 1).

Another approach to detect adverse effects is to aggregate individual adverse effects reports using the *OpenFDA* data set. This data set can then be used to predict possible adverse effects or drug interactions not mentioned in the drugs' labels as it is done in Bohm et al. (2016). However, the authors indicate that "[f]alse negative and false positive warnings frequently occur" (Bohm et al., 2016, p. 13) and that it is therefore important to also use other methods to detect drug interactions.

4.3.4 Clinical Trials

Several information sources for clinical trials were identified (Tab. 4.2). Although no service provides an API to access its data, all of them have the possibility to manually download a XML file containing all entries of a search.

As the clinical trials' location plays a vital role in assessing their relevance, a data source is needed that is not solely focused on one or a few countries. As the WHO registry platform seems to aggregate data from many national registers and other services, the choice would most likely fall on the WHO service.

4.3.5 Medical News

Many online services offer medical news, e.g. *MedScape*, *ScienceDaily* or *Medical News Today*. Usually their content is not accessible over an API and there can

[1] A boxed warning is a special section on a drug's package insert or label that resembles the strongest warning type of a drug's adverse effects, contra indications or interactions.

Name	Description	API	Access	Type	Country
ClinicalTrials.gov	Trial registry from US National Institute of Health. 39% are U.S. only trials.	no	Public & free	Register	Worldwide with a focus on U.S. (39%)
EU Clinical Trials Register	Clinical Trials Register for trials in the EU	no	Public & free	Register	European Union
German Clinical Trials Register	German clinical trial register. Also imports trials from clincaltrials.gov that are located in Germany	no	Public & free	Register	Germany
WHO International Clinical Trials Registry Platform	Search portal to central database with links to original records. Regular fetch of trials from currently 16 data providers, including sources mentioned earlier	no	Public & free	Search Service	Worldwide

Table 4.2: Identified data sources for the clinical trial locator

also be legal issues prohibiting any "unauthorized copy, reproduction, distribution, publication, display, modification, or transmission of any part of this Service" (ScienceDaily, 2016). An alternative to this might be the online social networking service *Twitter* which is used by many medical news sites, conferences and medical professionals to publish and promote recent findings and other information. An advantage of using *Twitter* might be the length limitations of 140 characters per "Tweet" which forces their users to focus on the most essential parts of what they want to say.

Chapter 5

Software Architecture

This section describes the prototypical implementation and integration of the CDSS into the existing EHR application described in the Introduction (Section 1.2). Not all of the previously mentioned information services can be fully implemented in the scope of this thesis. This is partly due to time constraints but is also attributed to the fact that high quality medical data sources are often commercial products that could not be gotten access to.

Section 5.1 provides an overview of the system architecture and its integration into the EHR application architecture. Then, two of the proposed clinical decision support services from Section 4.2 are implemented with the *Literature Service* in Section 5.2 and the drug interactions from the *Drug Information Service* in Section 5.3. Section 5.4 introduces the *Decision Support Controller* which resembles the API between client and business logic. Finally, Section 5.5 describes the user interface's architecture.

5.1 System Overview

The application is organised in a three layer architecture where each layer consists of components that encapsulate logically separable units. The proposed CDSS will be implemented as an own module in the business logic layer as well as in the client layer of an already existing EHR application (Figure 5.1). Existing components are

surrounded by dashed lines while newly added components are rendered with solid borders.

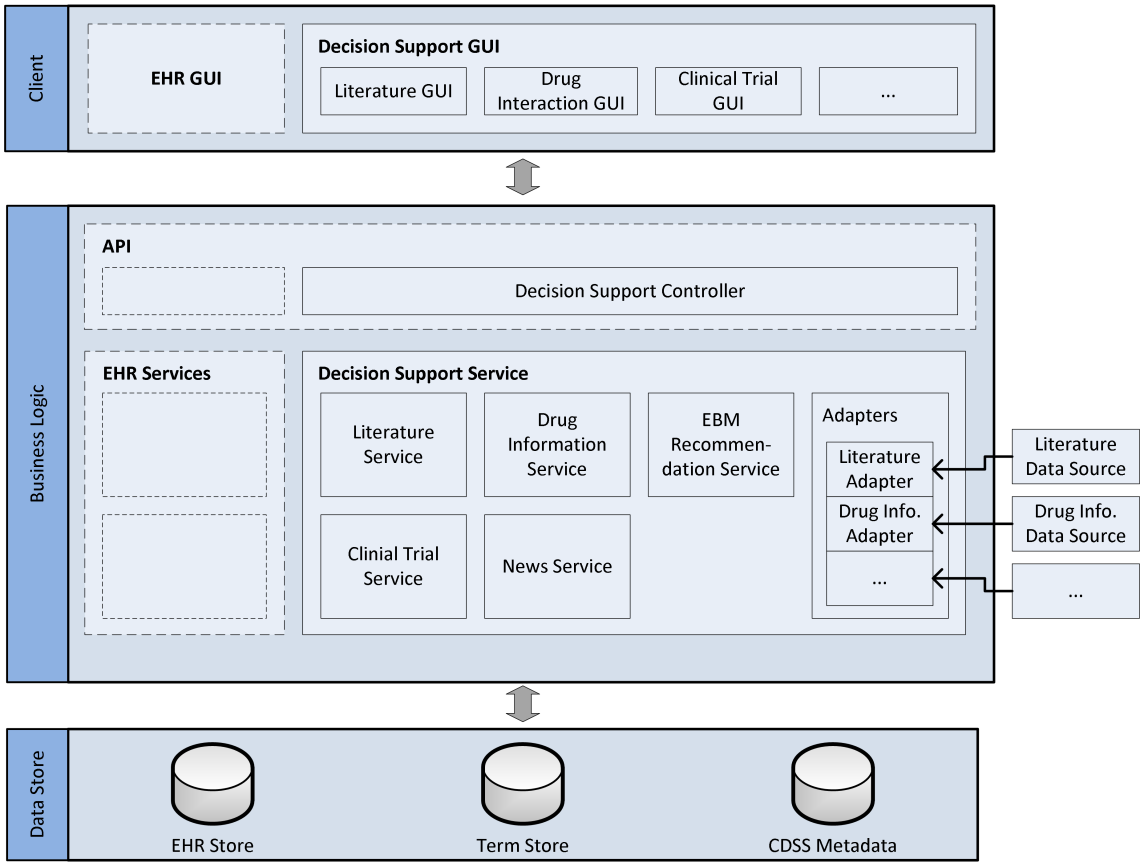

Figure 5.1: Three-layer system architecture

The different CDSS services like the literature or drug information service are sub-components of the main decision support service module. To communicate with the client decision support GUI, a new controller is added to the existing API module. The data sources displayed on the right side are not part of the system but are third party data sources that can be accessed over adapters. These adapters can be used by any decision support service. CDSS specific data is persisted in the new `CDSS Metadata` store. The `EHR Store` as well as the `Term Store` already exist in the current application and both of them are used by decision support services. The first allows storing EHR data in a standardised format and accessing the data over an object relational mapper (ORM) tool (Humm et al., 2015). The latter is used for storing an ontology for providing semantic autosuggest in the EHR and is implemented using the open source search platform *Apache Solr* (Beez et al., 2015).

Similar to the business logic layer, a separate CDSS module that is implemented alongside the EHR GUI encapsulates all CDSS logic on the client. The different service panels are all subcomponents of this module. This enables the display of the complete CDSS at any place in the EHR GUI.

5.2 Literature Service

To provide physicians with relevant (Requirement 1.1) and personalised (Requirement 1.2) information, a *literature service* was proposed in Section 4.2.1. This section describes the architecture and business logic this CDSS service in more detail.

5.2.1 Overview

The literature service takes an EHR as the basis for query generation, issues this generated query against a literature data source, retrieves the found literature and processes it to allow a user-friendly presentation and improve retrieval quality. To accomplish this task the service leverages an external dependency, the literature data source.

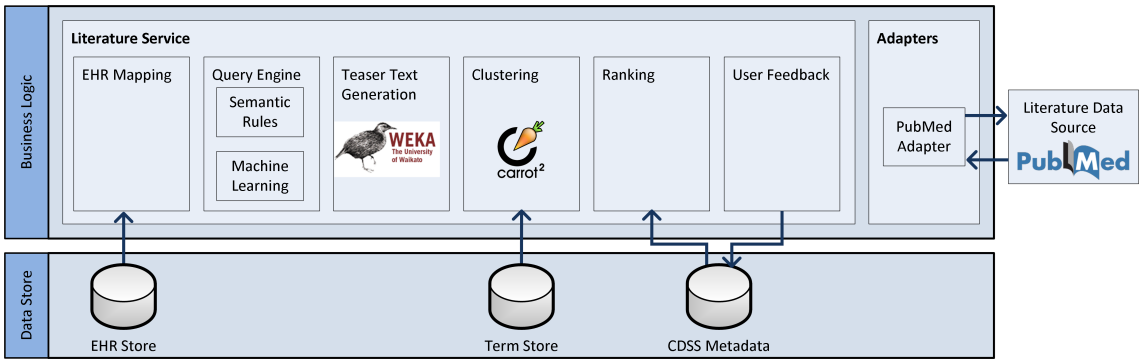

Figure 5.2: Literature service system architecture

After a request to the API controller is made by the client, the entry point in the literature service is the *EHR Mapping* component (Figure 5.2). The correct EHR is fetched from the *EHR Store* and translated into a literature service internal representation during *EHR Mapping*. This internal representation is the basis for the creation of a query by applying two strategies, semantic rules as well as a machine

learning (ML) approach. The final query is sent to the literature data source and the response is translated into an internal representation using the respective *Adapter*. For each found literature, a teaser text is automatically generated using ML. In order to provide users with a way to quickly navigate the found publications, the publications are clustered and appropriate cluster labels are automatically generated. By filtering the generated cluster labels using the *Term Store*, a high cluster label quality is achieved. The literature is then ranked by using stored user feedback as well as similarity measures. Once displayed, users can give feedback on the usefulness and relevance of the found literature. This feedback is stored in the *CDSS Metadata* store.

5.2.2 Data Source Selection

As outlined in Section 4.3.1, several potential data sources were identified that could be used for the literature service. Well-known and trusted among physicians and medical students is the search engine *PubMed* (Maggio et al., 2013, p. 2016). Its extensive collection of around 24.6 million available records of which all abstracts are publicly and freely searchable over an API, makes it well suited for integrating it into an application. All the publications are manually indexed with medical subject headings (MeSH) terms. MeSH is a controlled vocabulary or thesaurus maintained by the U.S. National Library of Medicine (NLM). The MeSH terms are also used for query expansion and allow to find publications that contain a synonym of the search term instead of the search term itself. Additionally, it includes publications from other databases like the *Cochrane Database of Systematic Reviews*. Although other databases like *Scopus* might have a seemingly bigger collection of articles, they often cover many scientific fields and are not limited to the health domain like *PubMed*. A study comparing several medical databases, including *PubMed*, *Google Scholar* and *Scopus* also states that "PubMed remains an optimal tool in biomedical electronic research" (Falagas et al., 2008, p. 1). Due to all of the above mentioned points, *PubMed* was selected as the data source for the literature service.

After selecting *PubMed* as data source, the question is wether to hold a copy of the data in a local search engine or to use the *PubMed* search engine over the online API each time a request is made (Table 5.1).

	Local search engine	PubMed online API
Pros	Always available	Low maintenance effort
	Customisation possible	Low implementation effort
	Potentially better query speed	Query expansion using MeSH
Cons	High implementation effort	Availability?
	Keeping data set current	No customisation of search
		Unwanted changes?

Table 5.1: Pros and cons of using a local search engine vs PubMed online API

The decision is basically a make-or-buy decision. As such, the local search engine solution has the typical pros and cons of a making a service instead of buying it. It is always available and can be customised to fit exactly the need but it also has a high implementation effort. Additionally, there are challenges in keeping the data set up-to-date with *PubMed* as publications can be updated, e.g. when MeSH terms are added. Querying *PubMed* takes around 4 to 8 seconds. That time might be shortened if a local search engine is used. However, as *PubMed* includes a large number of entries, appropriate hardware would be needed to store and query these.

Using *PubMed* as the search engine on the other hand provides an already implemented solution with low maintenance and implementation effort. It features a vector space retrieval model and provides query expansion by using a controlled vocabulary. Arguments speaking against *PubMed* include the dependency from the service provider, especially considering the availability of the service and dependency on the provider not making any unwanted changes. As *PubMed* is run by the U.S. government institute NLM and is a trusted source for many physicians around the world, there seems to be little risk for this. The availability of the service seems to be good, however during the implementation phase the service was not usable one time.

In conclusion, as using a local solution will not provide significant benefits over using the online solution, using*PubMed* as search engine over the API is favoured and selected for this thesis. If later requirements or changes in the *PubMed* services make it necessary to switch to another solution, only a new *Adapter* will be necessary to include the new data source.

5.2.3 EHR Mapping

Although the EHR is already stored in a structured way, it is first mapped to an internal service representation in order to prepare the EHR for the use in a query. From ca. 100 attributes that are currently used in the EHR application, not all are helpful for getting personalised literature suggestions for a concrete patient. Fields with no relevance like the patient's name or procedures undertaken several years ago are omitted in the EHR mapping. Other fields, like the clinical stage or the ulceration field are modified during mapping. This filtering and modifying of the individual EHR attributes is done by converting their values into `SearchField` objects (Figure 5.3).

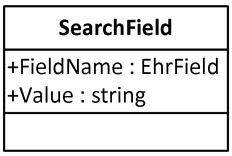

Figure 5.3: SearchField class

A `SearchField` object consists of a `FieldName` and the `Value`. The first indicates the EHR attribute the object was generated from, the latter holds the actual string representation of the EHR attribute. This value is later used to build the query. Examples for this mapping are the clinical stage and the ulceration field. As the clinical stage is just an enumeration consisting of roman numbers like "II" or "IV", the term "Stage" is added to the generated `SearchField`'s value. Similarly, the ulceration field is stored as a boolean attribute in the EHR and a `SearchField` containing the value "ulceration" is only created if the EHR attribute's value is "true".

Translation of an EHR into an internal representation happens on each literature service invocation in the `EhrMapper`. It is only invoked once per service call and stores the `SearchField` list for later usage. The mapper provides a method to get all terms identifying an EHR as well as one to get the values of a single EHR fields. For example, getting the values for the EHR field "Medication" might return a list of strings with the values "Warfarin" and "Ipilimumab". The `EhrMapper` is passed to various other components in subsequent processing steps and enables access to the EHR terms if needed.

5.2.4 Query Generation Engine

As *PubMed* is selected as the source for the literature service, a *PubMed* query is needed to find and display publications. To generate the query from the EHR, two strategies are employed: use of *semantic rules* for creating queries from EHR attributes and *machine learning*.

Semantic Rules

As mentioned earlier, non-relevant fields like the name or past procedures are already ignored during EHR mapping. However, other fields like the issue type, prescribed medications or current comorbidities are likely to return relevant results and are therefore included in the query by using semantic rules (Figure 5.4). Note that only a few sample rules are shown in the figure.

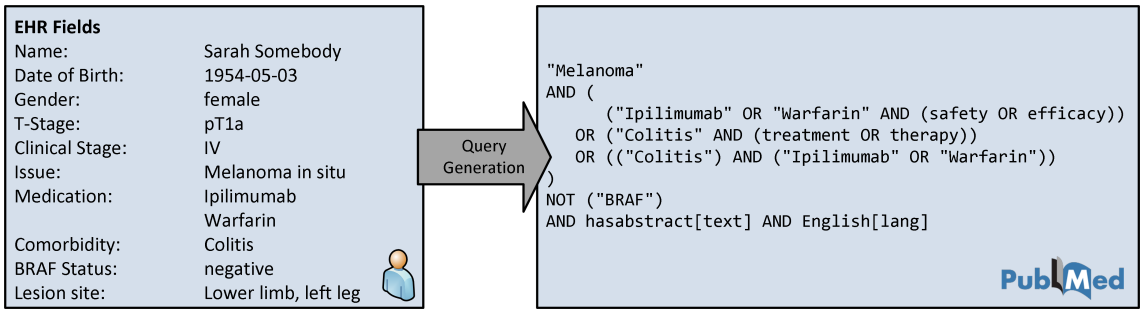

Figure 5.4: Sample query generation from an EHR using semantic rules

Rules try to meaningfully combine different EHR fields by combining all values of an EHR field with an `OR` and adding additional search terms to the subquery. A rule searching for publications that address the safety or efficacy aspects of the prescribed medications from Figure 5.4 would therefore result in the following subquery:

```
1 ( (Ipilimumab OR Warfarin) AND (safety OR efficacy) )
```

Listing 5.1: Sample medication subquery generated by a rule

Another rule combines the comorbidities field with the medication field to search for drug-related adverse effects and their treatment. If an EHR field's value consist of more than one word, it is surrounded by quotes to search for the specific term

and not the individual words. The basis for these rules are sample medical cases for which physicians documented their search process and the found relevant literature. By letting more physicians give feedback on other sample cases, additional rules could be identified and added to the query generation.

To ensure data quality and only search for recent literature, restrictions are added to the query like the "hasabstract[text]" to only show publications that contain an abstract. Currently, the following restrictions are added:

- `hasabstract[text]`: Only search for literature that has an abstract.
- `English[lang]`: Only search for English literature.
- `NOT letter[ptyp]`: As sometimes letters are retrieved that hold no medical evidence, ignore those.
- `humans[MeSH Terms]`: As *PubMed* also includes research on animals, this filer ensures that only literature applicable on humans is retrieved.
- `2011/09/23[PDat] : 2016/09/23[PDat]`: Limits the time scope of the returned literature as recent publications are the relevant ones.

Machine Learning Approach

The presented rule-based approach returns good results in various cases as fields like comorbidity, medication and issue are often relevant. However, in using a rules approach only a predefined field of possible questions can be answered and other fields like age, gender or the clinical stage could also in certain contexts return relevant literature. Additionally, the above mentioned terms "safety" and "efficacy" are only one example of additional query terms and other terms that are not captured in the semantic rules could also become relevant. Another feature possibly becoming relevant in the future might the preference of a specific publication type. Therefore, a query expansion method is applied by automatically refining queries based on user feedback. This is done using a machine learning approach.

Training The goal is to predict from the specific EHR which search terms retrieve the most relevant and helpful publications. Therefore, the EHR resembles the ML input whereas the potential search terms are the ML output to predict (Figure 5.5).

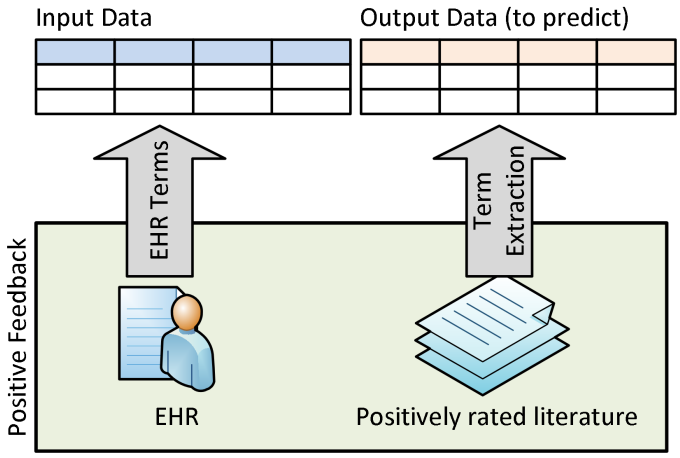

Figure 5.5: Training data creation from positive feedback

Training data for the machine learning is created from positive user feedback on the relevance and helpfulness of individual publications for a specific EHR (see Section 5.2.9). As input vector a BoW model is built from all EHR terms in the stored positive feedback. Potential search terms for the ML output, are acquired by extracting terms from the positively rated literature. This is done by leveraging the EHR application's medical ontology, the *Term Store*. All ontology terms that could be identified in the literature are added to the list of potential search terms. Additionally, the literature's MeSH terms and all EHR field value's appearing in the literature's title or abstract are also added to the output list.

The resulting ML problem is a multi-label classification problem as for each input instance a set of outputs labels have to be predicted. This is where a SVM is selected as it can provide multi-label classification and can deal with the high dimensionality of the input and output.

Training is currently an offline step due to possibly long training times and the fact that training only makes sense if there is enough training data, i.e. feedback, available. The generated ML model as well as the input BoW and the output BoW models are stored for later querying.

Querying For querying, the trained ML model is loaded and used to predict search terms for an EHR. For that, the current EHR terms are again transformed into an input vector using the stored input BoW model. As the BoW model only contains terms from already known EHRs, potentially new EHR terms not yet appearing

in the input BoW are ignored until another training iteration. After predicting the output terms, they are then used to search for literature. Both querying strategies are used in conjunction and the generated queries are concatenated by an `OR`. The final query is passed to the literature data source adapter which is described in more detail in the next section.

5.2.5 Literature Retrieval

Leveraging the `PubMedAdapter`, the previously generated query is sent to the *PubMed* search engine. The literature retrieval from *PubMed* is a two-step process where in the first request a list of PubMed Ids (PMID) is returned. The second request then utilises this list to fetch all found publications, their abstracts and other available information in one XML document. By providing the correct sort order parameter, the found literature is returned sorted by relevance. This means that publications being more similar to the query are ranked higher. More on *PubMed*'s relevance ranking in Section 5.2.8.

The XML results returned by the *PubMed* API are parsed into an internal literature object representation. From each literature in the XML, a literature object is created by only extracting fields needed in subsequent processing. Figure 5.6 shows the `Literature` class' attributes.

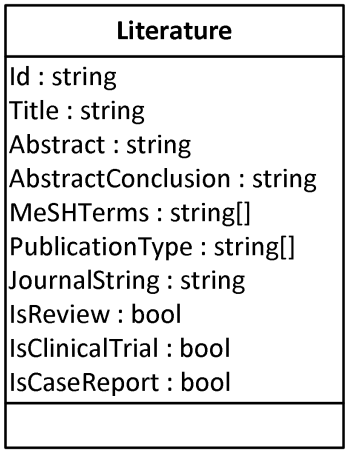

Figure 5.6: The literature class (UML)

As `Id`, the literature's PMID is used. The literature's title and abstract are adopted without changes and stored as strings. If the abstract has a section specifying its

conclusion, it is saved in an extra field. `MeSHTerms` used for indexing a publication are stored as a collection of strings. As the literature's publication type describes what kind of publication it is, e.g. a review, clinical trial or a journal article and a literature can have multiple publication types, they are also saved as a collection of strings. To display the different publication types in the client, boolean properties are introduced for the most important types. In order to display the publication year and journal information like the name, volume and issue, those XML fields are combined into the property `JournalString`.

5.2.6 Teaser Text Generation

If a publication title catches a user's attention, he may choose to read a teaser text in order to easily assess its relevance. As an abstract usually consists of 500 or more words, displaying that will not provide any benefit over visiting the actual *PubMed* site. Providing the conclusion of an abstract will better help in assessing the relevance of a literature. Abstracts that are structured, i.e. have special sections containing an introduction, background, results and a conclusion, can easily be used to display the relevant sections to the CDSS user. However, not all abstracts in *PubMed* are structured, about half of them consist of unstructured text. Therefore, for publications for which the conclusion is not explicitly marked, a machine learning algorithm is employed to predict the concluding sentences.

Training

To generate ML training data, the query in Listing 5.2 is used to retrieve all melanoma-related publications from *PubMed*. From the 3,266 returned publications with abstracts, 1,542 have a structured abstract and 1,725 an unstructured one. Only the structured abstracts are used for ML training.

```
1 (melanoma[MeSH Terms] OR melanoma[All Fields])
2 AND (  Clinical Trial[ptyp]
3    AND hasabstract[text]
4    AND humans[MeSH Terms]
5    AND English[lang] )
```

Listing 5.2: PubMed query to retrieve all melanoma-related publications

As the goal is to determine if a sentence belongs to the conclusion or not, the abstracts are split into individual sentences. Sentences that belong to the conclusion of an abstract are marked as such. The individual sentences resemble the training instances from which a BoW model is built. The ML goal is to classify each instance into one of two classes, whether the instance is a concluding sentence or not. Similar to the query term prediction the ML training is an offline step.

Querying

The generated model is used during parsing of the PubMed-returned XML to `Literature` objects. The unstructured abstracts are split into sentences and for each sentence an input vector is created using the BoW model created during training. Utilising the trained model, the label for each input vector is predicted. The teaser text is created by combining all sentences of an abstract that were predicted as belonging to the conclusion.

5.2.7 Clustering

Different users are interested in different topics and want to have different questions answered. A one-fit-for-all ranking is therefore not sufficient. To accommodate this, the result set is clustered to allow filtering the publications according to different semantic criteria. Example filter labels include "Ipilimumab-induced Colitis", "Adverse Effects" and "Overall Survival". For clustering the publications, the open source search engine clustering server *Carrot2*[1] is used. It utilises specialised clustering algorithms that automatically create meaningful cluster labels from the publications' titles and abstracts.

Algorithm Selection

Carrot2 comes with three clustering algorithms, each having its own pros and cons. Table 5.2 gives an overview of the three algorithms. As the aim of the clustering approach is to get a better insight of the returned literature set, a high cluster diversity is beneficial. Also, the better the cluster labels describe their documents,

[1] `http://project.carrot2.org/`

Feature	Lingo	STC	k-means
Cluster diversity	High, many small (outlier) clusters highlighted	Low, small (outlier) clusters rarely highlighted	Low, small (outlier) clusters rarely highlighted
Cluster labels	Longer, often more descriptive	Shorter, but still appropriate	One-word only, may not always describe all documents in the cluster
Scalability	Low. For more than about 1000 documents, Lingo clustering will take a long time and large memory.	High	Low, based on similar data structures as Lingo.

Table 5.2: Characteristics of Lingo and STC clustering algorithms (Osiński et al., 2016)

the better the user can browse the literature set. As scalability or the clustering speed is not that big of an issue with the relative small number of literature, the clustering algorithm *Lingo* is selected (Osiński et al., 2004).

Preprocessing

Prior to sending the found literature's titles and abstracts to the clustering algorithm, some preprocessing is needed to ensure good cluster quality. Preprocessing includes the removal of numbers, unnecessary terms and parentheses. Not all numbers occurring in an abstract are removed. If they are part of a term, as in "anti-CTLA4 antibody" they are kept in the text as they hold some meaning that is necessary to understand the potentially created cluster label. Other terms or abbreviations introducing noise are also stripped from the text, e.g. measuring units like "mg/kg" or common terms with little benefit in meaning like "years" or "cells". Lastly, all parentheses and their enclosed contents are removed.

Clustering

The preprocessed abstracts are fed into clustering algorithm and results are parsed to a list of `Cluster` objects. Their class is described in Figure 5.7.

Each `Cluster` is identified over an Id. The `Score` indicates how good the algorithm estimates the cluster's quality. The `Phrases` property contains the assigned cluster

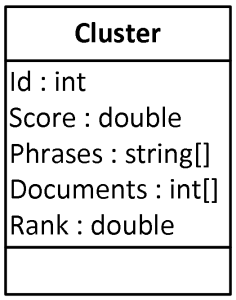

Figure 5.7: UML of the class Cluster

label and `Documents` contains the Ids of the the cluster's documents. The `Rank` property is not filled during clustering but during ranking of results (Section 5.2.8).

Cluster Label Filtering

As generated cluster labels may not always be useful, a filtering approach is applied using the EHR application's medical ontology by querying the *Term Store*. Generated cluster labels are split into individual words and checked against the *TermStore*. If at least one word of a cluster label is found in the ontology, the cluster label is accepted, otherwise the whole cluster is discarded. To further fine tune the cluster label filtering, a custom blacklist as well as a whitelist are added to the filtering process. Occurrences of cluster label words in the whitelist are treated as if they were found in the ontology and cluster labels consisting solely of terms found in the blacklist are discarded. The custom whitelist contains terms like "survival", "adverse" or "therapy" that were manually detected and considered as helpful terms occurring in cluster labels. Additionally, the list of terms describing the EHR are added to it.

5.2.8 Result Ranking

Although there is an alternative way of browsing the literature set with the created clusters and their labels described in the previous section, ranking of the literature is still important. The most relevant literature should hereby be ranked higher than the less important ones. Additionally, the generated cluster labels used for filtering (Section 5.2.7) also have to be sorted. Ranking is employed by leveraging *PubMed*'s

weighted relevance score while also incorporating user feedback on the relevance and quality of publications described in Section 5.2.9.

PubMed Rank

Since October 2013, *PubMed* provides the possibility to sort search results by relevance. Hereby, a score is calculated for each literature depending on how many search terms from the query are found in the literature and in which fields they appear (NLM Tech Bull, 2013). *PubMed* uses several index fields for calculating the relevance scores, e.g. the title, abstract and MeSH terms. The relevance score calculation for a multi-term query is based on the following measures (NCBI, 2016):

- IDF_{fw}: The global weight of a term $_w$ regarding the result set and index field $_f$
- TF_{fw}: The local weight or term frequency of a term $_w$ in a specific field $_f$.
- W_f: The weight of the field $_f$, e.g. a term occurring in the title field would have more importance than it occurring in the abstract.
- PW: The weight of the literature's publication date. More recent publications get a higher weight than older ones.

The score for each literature in the result set is calculated by summing for each query term $_w$ in each index field $_f$ and multiplying the result by PW, the weight of the literature's publication date (NCBI, 2016):

$$IDF_{fw} \times TF_{fw} \times W_f$$

By using the relevance sort option while retrieving literature from *PubMed*, the publications most similar to the query are at the top of the query result, with a slight preference of current publications. As *PubMed* does not provide the exact numbers of the relevance score calculations, a rank is built based on the sort order of the found literature: the publication retrieved first is assigned the rank 1, the second one the rank 2, the third one the rank 3 and so on.

Feedback Rank

Additionally to the use of the relevance-based rank, collected user feedback is added to the overall rank calculation. Using the stored feedback users may give on displayed literature in an EHR, three different literature rankings are generated based on the number and type of feedback a literature received. Feedback types consist of negative feedback, positive feedback and passive feedback (Section 5.2.9) and ranking is done by sorting the literature according to the number of feedback each has received, subsequently building a rank with the highest feedback literature having rank number 1.

Literature Service Rank Calculation

All of the previously mentioned ranks have to be merged together to form a new ranking that can be used for displaying the found literature to the user. Agichtein et al. (2006) mention a ranking approach where they ignore the original rankers scores (i.e. the *PubMed* TF*IDF score) and instead just merge the rank orders to integrate implicit feedback into the ranking. Similar to that approach, the four rankings described earlier are combined into one merged score S_M using the equation in 5.1 with ranks R and weights W:

$$\left.\begin{aligned} S_M = \quad & \frac{1}{R_{PubMed}+1} \\ & + \frac{1}{R_{positive}+1} \times W_{positive} \\ & + \frac{1}{R_{passive}+1} \times W_{passive} \\ & - \frac{1}{R_{negative}+1} \times W_{negative} \end{aligned}\right\} = UserFeedback \tag{5.1}$$

Apart from the negative feedback, all feedback and *PubMed* ranks are summed up. Negative feedback is subtracted from the sum. Each incorporated ranking is multiplied with the specific ranking weight. The weights are heuristically tuned and represent the relative importance of the feedback type. The found literature is sorted descending by the calculated merged score S_M.

Not only the found literature itself has to be ranked, the generated cluster labels also have to be sorted in order to display them to the user. Hence, the cluster

ranks are calculated by building the average rank of their contained documents and multiplying that with the *Carrot2* cluster score. The latter is a measure provided by the clustering algorithm that states how good the algorithm thinks the generated label is.

5.2.9 User Feedback

There are two kinds of user feedback gathered in this application, active and passive. Active feedback is generated by the users clicking on thumbs-up or thumbs-down icons in the client. Passive feedback is gathered by logging and interpreting all clicks on a publication. All feedback is stored in the CDSS Metadata store and is used for the ranking of publications and clusters (Section 5.2.8) as well as to create training data for machine learning (Section 5.2.4). All feedback is stored using one class that holds positive, negative as well as passive feedback (Figure 5.8).

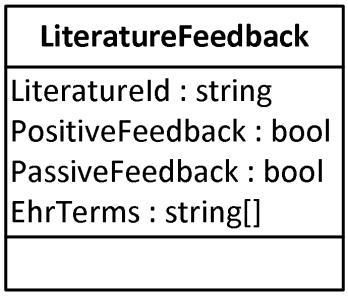

Figure 5.8: Literature feedback class

One can distinguish between the different forms feedback by looking at the two boolean attributes. The attribute "PositiveFeedback" holds both the active feedback types, i.e. positive and negative feedback. For identifying the feedback object, the unique Id of the voted literature is stored. As EHRs change over time and a feedback is always specific to an EHR at a certain point in time, the terms describing the EHR are also saved upon user feedback. For saving user feedback and filling the `EhrTerms` property, the user feedback component utilises the `EhrMapper` to convert an EHR to the list of EHR terms.

5.2.10 User Interface

The literature service displays the found literature and the generated clusters to the user. To not confuse the user, clusters are called filters as this term is better known among users (Figure 5.9). The red numbers are not party of the interface but are used to describe the figure.

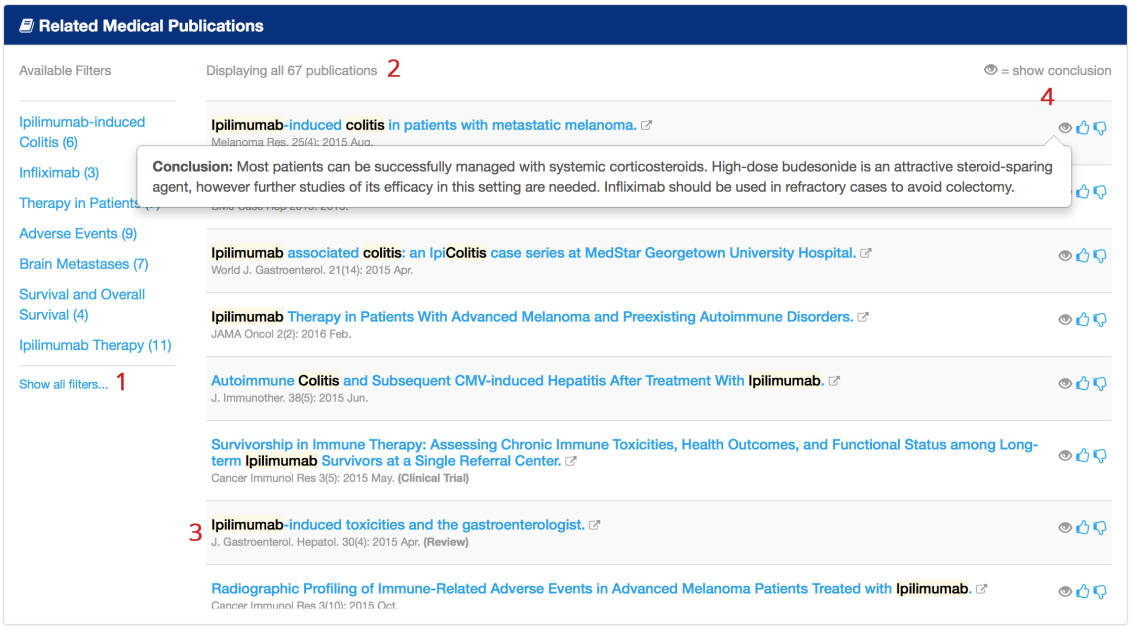

Figure 5.9: Literature service user interface

No filter is initially selected and only 7 filters are displayed, the rest is available over clicking the "Show all filter" link (1). On clicking a filter, the filter is rendered bold, the displayed literature filtered accordingly and a link to clear the filter is displayed behind the status message (2). The status message indicates how many publications are currently displayed. On clicking the link to clear the filter, the initial literature set is shown again. A found literature is displayed with its title and journal information, including the publication year (3). The title links to the literature's *PubMed* page and the light grey symbol behind the title indicates the link leading to an external website. The previously mentioned special publication types like "Clinical Trial" or "Review" are shown in bold behind the journal information. EHR terms occurring in a literature's title or teaser text are marked for quick reference. Teaser texts can be displayed by hovering over the eye icon and a legend indicates the function of this eye icon to the user (4). To give users the possibility to give feedback, each publication

features thumbs-up and thumbs-down icons. On clicking them, they change style and become filled and the feedback is send back to the server.

5.3 Drug Interaction Service

The drug interaction service is part of the greater drug information service which provides drug information at the point of care (Section 4.2.3). The drug interaction service checks a patient's prescribed medications but also additionally added ones by the user for interaction's among. Figure 5.10 shows the architecture of this drug interaction service.

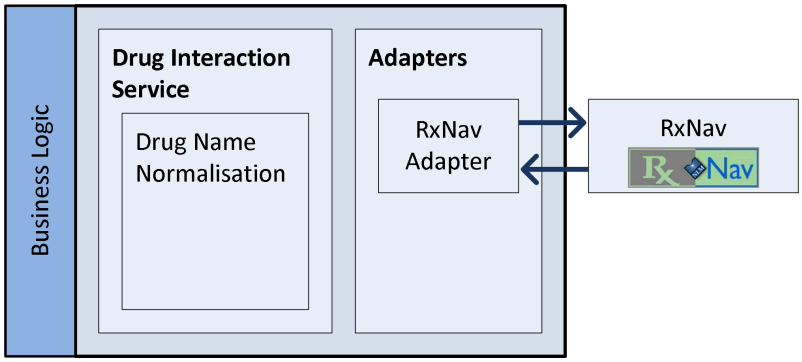

Figure 5.10: Drug interaction architecture

A drug interaction search starts automatically in the background once a user opens a patient's EHR in the client. A request is sent to the *Decision Support Controller* which fetches the patient's medication and invokes the *Drug Interaction Service* with a list of drug names. The drug interaction services uses this list and queries the data source before returning a collection of `DrugInteraction` objects to the client. The `DrugInteraction` class contains the interacting drugs, a description of the interaction and its severity (Figure 5.11).

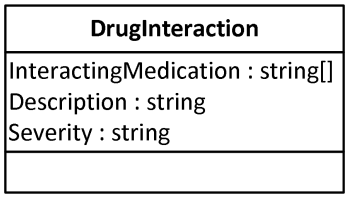

Figure 5.11: DrugInteraction class

5.3.1 Data Source Selection

Section 4.3.3 and Appendix C listed and discussed different sources for drug information data, including sources to acquire drug interaction data. As it is not possible in this thesis to use commercial data sources, only six of the identified drug information sources remain: *Drugs.com*, *Electronic Medicines Compendium*, *MedScape*, *DailyMed*, *DrugBank* and *RxNav*. As *Drugs.com* prohibits its integration into any kind of IR system and *Electronic Medicines Compendium* as well as *MedScape* do not provide an API and only contain natural language texts, three services remain. Although *DailyMed* contains current package inserts of marketed drugs in the U.S. and also provides an API, the specific drug interaction information is still in a natural language form. This leaves *RxNav* as *DrugBank* is queried directly by *RxNav*. An API allows easy integration into the application and the included *RxNorm* service normalises drug names between different drug dictionaries. This is an advantage as other drug information sources using different drug dictionaries could be added in the future. Unfortunately, *DrugBank* and therefore also *RxNav* does not provide any information on the severity of drug interactions. As the severity is important for the usability of and trust in a CDSS system, the selection of a commercial data sources would most likely be preferable for a live EHR system. However, for the sake of the prototypical implementation, *RxNav* is chosen as data source.

5.3.2 Drug Interaction Search

Apart from the earlier mentioned automatic search for interactions among prescribed medications, the user also has the possibility to search for additional drugs and their potential interactions with the prescribed ones. The actual search is not different to the initial one and consists of two steps. First, the supplied drug names or drug ingredients have to be translated into a form that can be used to query the drug interaction data source. Therefore, the drug normalisation service *RxNorm* is used to search for the unique drug identifier *RxCui*. If a drug could not be resolved to an *RxCui*, nothing is returned and the drug is therefore not included in the drug interaction checking. Second, the generated list of *RxCui*s is used to query for interactions using the *RxNav* API. The API's result is parsed back to a list of `DrugInteraction`-objects and returned to the client for displaying panels and alerts.

5.3.3 Drug Interaction Interface

The drug interaction interface allows users to add additional medication to the interaction checking and see details about the found drug interactions (Figure 5.12).

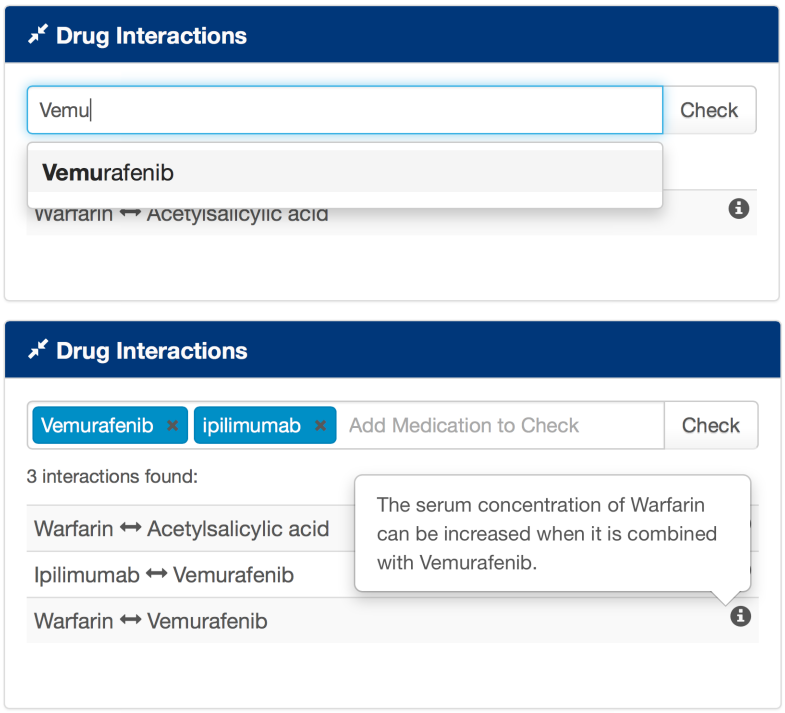

Figure 5.12: Drug interaction panel

Initially, only the drugs currently prescribed are checked and displayed in the panel. The user has the option to enter additional drugs, for example the ones that are planned to be prescribed. To aid the user, the EHR application's ontology is queried and possible matches displayed in an autosuggest box (Figure 5.12, top). On selecting an option, the search is automatically started again. The user can however also initiate the interaction search over the "Check" button. Additional drugs are rendered in a box that also features an icon to remove the specific drug from the interaction checking. Found interactions are displayed in a list and hovering the information icon on the right provides more information of the interaction.

In addition to the drug interaction panel, severe drug interactions should be shown as a well visible alert. Unfortunately, as the selected data source does not provide severity data for its interactions, for now all found interactions are displayed.

5.4 Decision Support Controller

All communication between the client layer and the business logic layer is routed through the EHR application's API. As each EHR service has its own API controller, the decision support system also features a `Decision Support Controller`. It is responsible for handling requests made by the client and calling the appropriate business logic, i.e. decision support services with the correct parameters. Currently, three different endpoints are handled by the controller:

- `SearchLiterature`: To search literature for a specific EHR, an EHR identifier has to be supplied in the request. Using that, the `Decision Support Controller` fetches the correct EHR with all depended fields from the database. The fetched EHR is then passed to the literature service and the results returned to the client.
- `LiteratureFeedback`: To save feedback on a literature, the EHR identifier and a `Literature Feedback` object (Figure 5.8) are supplied in the client's request. After fetching the EHR, the feedback object is sent to the literature service's user feedback component (Section 5.2.9) to process and store the feedback. A response indicating the correct saving of the feedback is returned to the client.
- `GetDrugInteractions`: The drug interactions endpoint takes a request holding the patient Id as well as an optional list of additional medication. Using the patient Id, the correct patient is retrieved from the database and its currently prescribed medications fetched. The drug interaction service is invoked by supplying the fetched medications as well as the provided additional medications list. The found list of drug interactions is returned back to the client.

5.5 GUI Architecture

Although most logic is implemented in the business logic layer, the client layer also needs some to ensure a good user interaction. The CDSS is realised as a complete Decision Support Module where each individual CDSS service represents its own subcomponent. Every module can send alerts or reminders over the alert bus which will be collected in the CDSS controller and displayed in the CDSS view. Other CDSS

subcomponents also consist of a controller and a view. The individual controllers are responsible for communicating with the correct API endpoints of the business logic layer. The single services could therefore be easily extracted from the *Decision Support Module* and displayed at any other point in the EHR.

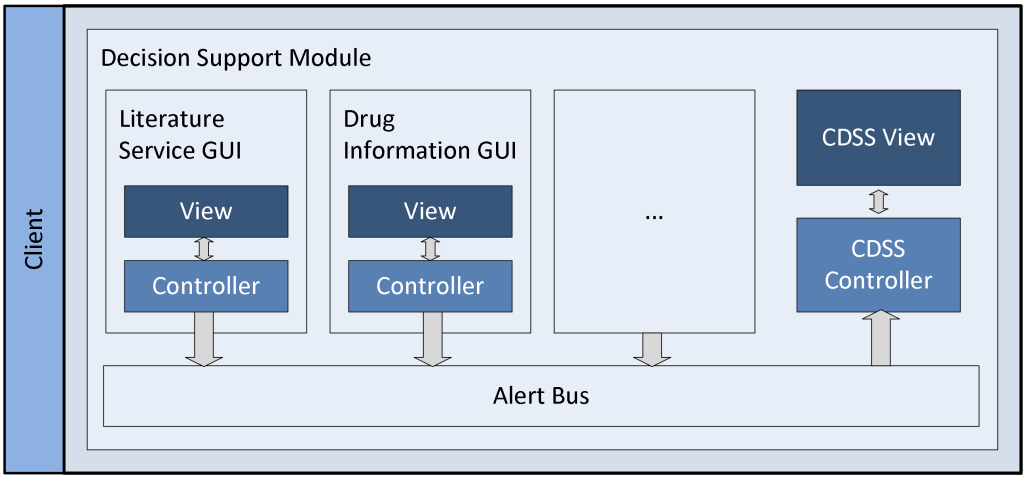

Figure 5.13: Client GUI architecture

Chapter 6

Implementation

The application is implemented in *C#* using .NET and MS SQL Server on the server side, and in HTML5 / CSS / JavaScript on the client side, using the frameworks *Bootstrap* and *AngularJS*. As this is already used in the development of the current EHR application, the CDSS also leverages this technologies.

Two of the proposed CDSS modules were implemented as part of this thesis. Section 6.1 describes the implementation of one of them, the literature service. The implementation of the drug interaction service is explained in more detail in Section 6.2.

6.1 Literature Service

Section 5.2 described a literature service that finds and retrieves relevant and helpful literature fitting a specific EHR. This section illustrates special implementation details of this service. Entry point into the literature search is the `EhrMapper` described in Section 5.2.3. It translates the EHR which is distributed over different database objects into a flat list of `SearchField` objects for easier handling in subsequent processing steps. A reference of this `EhrMapper` is given to the the `QueryGenerator`.

6.1.1 Query Generation Using Rules

Section 5.2.4 proposed the idea of using rules to query for relevant literature. The `QueryGenerator` implements this functionality and takes an `EhrMapper` as input. To allow dynamic building of rules, a composite pattern is introduced (Figure 6.1) that enables the creation of a query expression tree.

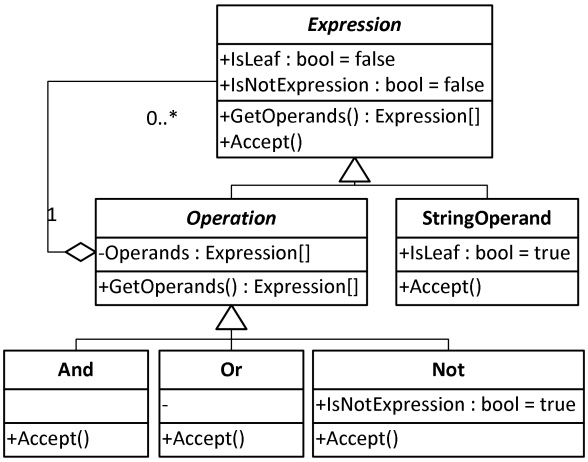

Figure 6.1: Query expression composite pattern

The composite pattern allows clients to treat individual objects and compositions uniformly. The abstract class `Expression` is extended and its abstract methods implemented by all subclasses. The `Operation` class contains a collection of Expression objects resembling the operands of a operation. The actual operation classes like `And`, `Or` and `Not` hold little additionally information and simply extend the `Operation` class. The `StringOperand` class does not have a collection of `Expression` objects and does therefore not have any descendants as it usually forms the leafs of an expression tree. To allow easy iteration of the trees, the `StringOperand` and the `Not` operation do have the appropriate boolean flags set to true, indicating their type. All classes feature a method called `Accept` that can be called by the `Iterator` and accepts the `Visitor`. The `Iterator` is responsible for traversing the expression tree whereas the `Visitor` builds the actual query by visiting the nodes and adding the appropriate query parts according the the visited node's class. Building an expression tree can be done manually as in Listing 6.1 or by using the `RuleQueryBuilder`.

The `RuleQueryBuilder` construct the expression tree by calling different methods. This enables a clear internal representation of rules and should allow to add new rules more quickly. The class provides three method types to add rules:

```
new And(
    new StringOperand("Ipilimumab"),
    new Or(
        new StringOperand("safety"),
        new StringOperand("efficacy")
    )
)
```

Listing 6.1: Query expression tree creation

- `AddRule`: Add a rule by providing the operation (`AND`, `OR`, `NOT`) and its operands. Operands can either be a list of `EhrFields` or a list of string search terms.
- `AddExtendedRule`: This allows to add a rule by providing the operation (`AND`, `OR`, `NOT`), the `EhrField` and additional search terms as strings.
- `AddRestriction`: Allows adding query restrictions that are added to the query with an `AND`, effectively reducing the result set as the term has to occur in each publication returned.

The `EhrFields` are fetched from the `EhrMapper` and the values of one field concatenated with an `OR`. On building the expression tree, all rules are concatenated with an "OR", effectively increasing the results while the restrictions are added with an "AND", effectively reducing the number of results. While getting the EhrField values from the `EhrMapper`, values containing more than one word are surrounded by quotes whereas the rest is usually added to the query without quotes. This enables the search engine *PubMed* to use query expansion on the non-quoted terms while at the same time ensuring that multi-word terms are searched for in whole. A sample rule and its output can be seen in Listing 6.2.

```
queryBuilder.AddExtendedRule(new And(), EhrField.Medication,
    ↪ "safety", "efficacy");
//Result:
//((Ipilimumab OR Warfarin) AND (safety OR efficacy))
```

Listing 6.2: Sample query rule creation

6.1.2 Predicting Search Terms Using Machine Learning

To facilitate machine learning and predict suitable search terms, *Accord.NET*'s[1] one-against-all multi label SVM is used. Basis for the training data is stored user feedback on the relevance of publications.

Training Mode

Training of the SVM is currently initiated over the test project. The actual training data is gathered from positive feedback which is iterated over. To generate the input vector, two lists are needed. One to hold all EHR terms occurring in all feedback instances and one to hold the EHR terms of the individual feedbacks. Each individual feedback resembles a training instance and could therefore be seen as the rows of the input vector. Figure 6.2 illustrates this for better understanding.

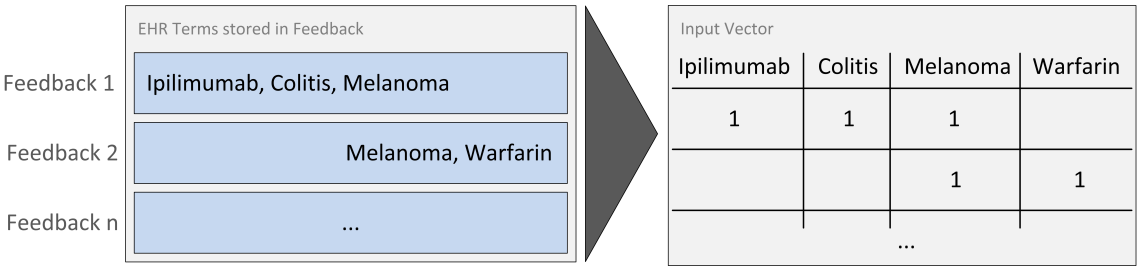

Figure 6.2: Sample conversion of feedback instances to an input vector

Listing 6.3 shows the actual implentation of creating the two lists and using them to create the input vector. First, all feedback is gathered from the database (line 1) and the two lists are initialised (line 2 and 3). Iterating over the `feedbackList` allows to fill the two lists accordingly. They are used to initialise a BoW object (line 16) that is subsequently used to populate the actual input vector (line 19 - 22). The double array `inputs` is then used in subsequent training of the SVM.

```
1 var feedbackList = FeedbackDao.FindAllFeedback();
2 var allEhrTerms = new HashSet<string>();
3 var feedbackEhrTerms = new List<List<string>>();
4 foreach (var feedback in feedbackList)
5 {
6     // Compute all features for the input
```

[1] http://accord-framework.net/

```
7       var tmpList = new List<string>();
8       foreach (var term in feedback.EhrTerms)
9       {
10          tmpList.Add(term.ToLower());
11          allEhrTerms.Add(term.ToLower());
12      }
13      feedbackEhrTerms.Add(tmpList);
14  }
15  var inputBagOfWords = new BagOfWords(allEhrTerms.ToArray());
16  var inputs = new double[feedbackList.Count][];
17  // Populate the inputs and outputs
18  for (var i = 0; i < feedbackList.Count; i++)
19  {
20      var inputArray = inputBagOfWords.GetFeatureVector(
            ↪ feedbackEhrTerms[i].ToArray());
21      inputs[i] = IntToDoubleArray(inputIntArray);
22  }
```

Listing 6.3: Data preparation to build input vector

Building the output vector is very similar to the construction of the input vector. However, the terms used to do so first have to be extracted from the positively voted literature. Therefore, the literature has to be fetched from *PubMed* again. As there is a high possibility that some publications occur several times in different feedbacks, a caching strategy is applied by filling a dictionary with the PubMed Id and the extracted terms of a literature. As a result, the term extraction has to be done only once per literature. The actual term extraction is bundled in the `TermExtraction`-class. The most important resource for terms describing a literature are a literature's MeSH terms. They are assigned by humans and usually provide a better quality than terms acquired using NLP tools. However, especially recent publications are not yet indexed with MeSH terms. Therefore, another approach is needed to extract terms describing the specific literature.

Using the open-source NLP library *OpenNLP*[2], the literature's title and abstract are split into sentences and then tokenised. The code displayed in Listing 6.4 iterates over those tokens and searches for matches in the EHR application's ontology by trying all possible token combination lengths. This code is part of the previous thesis by Beez (2015).

[2] `https://github.com/AlexPoint/OpenNlp` Note: This is not officially affiliated to the Java-based Apache OpenNLP toolkit.

```
1  var returnValue = new List<Term>();
2  while (tokens.Count > 0)
3  {
4      Term currentTerm = null;
5      for (var i = 1; i <= tokens.Count; i++)
6      {
7          var searchFor = StringFromList(0, tokens, i);
8          var foundTerms =
               ↪ TerminologyService.QueryAllCategories(searchFor);
9          var term = Find(searchFor, foundTerms);
10         if (term != null) currentTerm = term;
11     }
12     if (currentTerm != null)
13     {
14         returnValue.Add(currentTerm);
15     }
16     tokens.RemoveAt(0);
17 }
18 return returnValue;
```

Listing 6.4: Identifying ontology terms (Beez, 2015)

As ontology terms often consist of more than one token, a search for each individual token might not yield the desired results. Therefore, the sentence's tokens are iterated over in a `while`-loop (line 2) and the first token in the collection removed after each iteration (line 16). Inside the loop, another loop iterates from 1 to the end of the token collection. Inside the `for`-loop, a string is built with the tokens from the beginning until the current iteration of the `for`-loop. This effectively allows to search for all possible token concatenations and is visualised in Figure 6.3 for clarity.

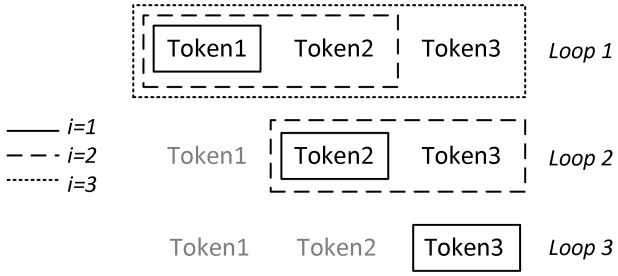

Figure 6.3: Algorithm illustration for identifying ontology terms

Taking a list of three tokens as an example, three possible token concatenations are possible in the first loop. The Token in the box with a solid line is one combination,

the tokens in the dashed box a second one and the tokens in the dotted box the third. Because after each `while`-iteration the first element of the token list is removed, the second `while`-loop starts with the second token. Each of the concatenated token combinations are used to query the EHR application's *TermStore* (line 8). As the *TermStore* also returns entries that contain more words than it was searched for, only entries that match exactly the search terms are further processed (line 9). As longer words should describe the literature more specific than short ones, the variable in line 4 is overridden if a new term is found.

In addition to the MeSH terms and the term extraction from abstracts and titles, all the EHR terms occurring in the literature are also added to the output term collection. This enables searching for EHR terms that are not used in the rule-based approach and would therefore be forgotten. Before returning the extracted terms used for the output vector, all duplicates are removed.

Once the input as well as the output vector are created, a multi-label SVM is initialised, trained and serialised as a binary file to disk using a `BinaryFormatter`. Alongside the trained SVM model, the input and the output BoW model is stored. They are later used during querying to translate an EHR into a feature vector to predict fitting search terms.

Query Mode

When search terms have to be predicted for an EHR, the serialised SVM model is deserialised into a C# object again. The input BoW model that was stored alongside the SVM is used to create a feature vector of the current EHR which is given to the SVM model for search term prediction. As the SVM returns predictions as a boolean array, the output BoW model is again used to transfer the boolean array into a list of search terms by iterating over the boolean array and comparing the array's indices with the BoW model's indices. The actual predicted search terms are added as a rule to the rule query by concatenating them with an `OR`.

6.1.3 Literature Retrieval from PubMed

After the query is created the literature has to be retrieved from *PubMed*. This is a two-step process as first the literature's PMIDs have to be searched using the

eSearch API and subsequently fetched using the *eFetch* API. All communication with *PubMed* happens over the adapter class `PubMedApi`.

Searching PMIDs

The first step is ususally to search for literature Ids over the *eSearch* API. Therefore, a HTTP request is prepared and issued against the API endpoint[3]. Several parameters are added to the request: the encoding, the database to query, the format in which the results should be returned, the query, the sort order and the number of results to return. When adding the query to the request, all spaces are replaced with a "+" and the return mode is set to "json". The "sort" parameter is set to return results in relevance based order. An optional parameter allows specifying the maximum result size. By default this value is set to 70. As indicated with the return mode parameter, the result is in JavaScript Object Notation (JSON) format and contains a list of PMIDs. The JSON is parsed and the PMIDs extracted using *Json.NET*[4]. The extracted PMIDs are subsequently used to fetch the literature and to identify the literature in the application.

Fetching Literature

The second step is retrieving and parsing literature over the *eFetch* API[5]. The method in the `PubMedAPI` class takes a list of PMIDs and queries the API endpoint. In contrast to the literature search, the HTTP request parameters are set to "XML" as return mode and "abstract" as return type. Instead of the query, the list of relevance-ordered PMIDs is provided as parameter. The returned XML is parsed into `Literature` objects (Figure 5.6) using the `PubMedXMLParser` class:

If the retrieved XML does not have a root node with the name "PubmedArticleSet", an error is thrown. As the individual publications are contained in that one XML, it is split into individual articles by iterating over all children of the root node. Each child node is processed using XML Path Language (XPath) which is built into C#. The values of the XML nodes PMID, article title, article abstract and if available its conclusion are directly assigned to the newly generated `Literature` object. The

[3] `https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi`

[4] `http://www.newtonsoft.com/json`

[5] `https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi`

`JournalString` is built by using different XML nodes like the journal title, journal volume, journal issue and the publication year. All MeSH terms are collected and only distinct values added to the `Literature` object. The same happens with the publication types. Additionally, if one of the types is either a clinical trial, case report or review, the `Literature` object's boolean variables `ClinicalTrial`, `CaseReport` or `Review` are set to `true`.

Literature Caching

To speed up the fetching process, the found literature is cached using .NET's *MemoryCache* class. Using the `PubMedCache` wrapper class, the fetched literature is added to cache with its PMIDs as identifier with a sliding expiration date of two days. This means that if in two days no request is made for a particular PMID, the literature is removed from cache. If however a request is made, the expiration date is updated. On subsequent literature fetch requests, the literature is first searched for in the cache and retrieved if found. All PMIDs that could not be found in cache are sent to PubMed to be retrieved and are then also added to the cache for later querying.

6.1.4 Teaser Text Generation Using Weka

After searching and retrieving literature from *PubMed*, teaser texts have to be generated. For already structured abstracts, the sections marked as conclusions are taken as teaser text. For unstructured abstracts a ML approach was proposed in Section 5.2.6.

Technology Selection

To implement this, the open-source machine learning software suite *Weka*[6] is employed. This selection was made for several reasons:

- *Preprocessing and ML algorithms.* It features a comprehensive collection of ML algorithms and preprocessing techniques.

[6] `http://www.cs.waikato.ac.nz/ml/weka/`

- *GUI*. The ML suite provides a GUI that allows easy and comfortable testing of different preprocessing strategies, ML algorithms and their parameters. This also allows to create a ML model using the GUI and use that model from within the application.

- *String2WordVector*. It already provides a filter to transform strings into word vectors which is needed to create the BoW model.

- *Sparse Vectors*. Especially in text processing the input vectors can grow exceptionally large which results in a lot of RAM being used. *Weka* provides a possibility to build and use sparse vectors[7] which results in considerably lower RAM consumption.

Weka is written in the *Java* programming language and thus runs on almost every modern operating system. However, it has to be integrated into the existing EHR application which is written in *C#*. The recommended way by the *Weka* team is to use *IKVM.NET*[8] in order to run *Weka*. *IKVM.NET* implements a Java Virtual Machine (JVM) in the Microsoft .NET Framework and also provides a .NET implementation of the Java class libraries. This allows to call the java-based *Weka* code directly from *C#*, while also providing the possibility to ship *Weka* as a project dependency. This can be achieved by compiling the *Weka* `.jar` file into a *Dynamic Link Library* (DLL) using the *IKVM* command `ikvmc <filename>.jar`.

Data Preparation

As already described, all abstracts have to be split into sentences. The structured ones only one time to build the training data and the unstructured ones each time the literature search is triggered. To split abstracts into sentences, a open-source NLP library is used[9]. However, only splitting the texts into sentences without further processing results in noisy data as conclusions are often the place where non-medical information is put. An example for this are clinical trials that have fragments like "ClinicalTrials.gov number, NCT01024231." in their conclusion. This is not desirable to show as a teaser text. Therefore, the sentences are tokenised and the resulting

[7] Sparse vectors use a special format where only the occurrence of a word is stored. This results in considerably smaller vectors.

[8] `https://www.ikvm.net/`

[9] `https://github.com/AlexPoint/OpenNlp`

tokens POS tagged. By removing all sentences without a Verb, the data quality can be drastically enhanced. Listing 6.5 outlines the code used to do the sentence splitting and filtering.

```
1  public IList<Sentence> SplitIntoSentences(string text)
2      string[] sentences = SplitSentences(text);
3      var processedSentences = new List<Sentence>();
4      var position = 0;
5      foreach (string sentence in sentences)
6      {
7          string[] tokens = TokenizeSentence(sentence);
8          string[] tags = PosTagTokens(tokens);
9          if (ContainsVerb(tags))
10         {
11             // Calculate relative position in Text
12             var newSentence = new Sentence(sentence);
13             newSentence.Position = (100 / sentences.Length) *
                   ↪ position++;
14
15             //Add sentence to return value
16             processedSentences.Add(newSentence);
17         }
18     }
19     return processedSentences;
20 }
```

Listing 6.5: Sentence splitting and cleansing using a POS tagger

Line 2 calls the NLP library's `EnglishMaximumEntropySentenceDetector` which splits the provided text into individual sentences. Iterating all of the sentences, each of them is tokenised and the tokens assigned their POS tags (line 7 & 8). If the sentence contains a verb, it is added to the return collection. Apart from the actual sentence text, the sentence's position in the abstract is also calculated and stored (line 13). In order to generate the training data it was also checked if the sentence belongs to the conclusion by comparing it against the actual structured conclusion text. As this was a one time offline step, it is not shown here.

The sentence splitting as well as the POS tagging need trained models to split sentences or assign POS tags. Unfortunatly, the POS tagger's API expects a string

containing the file path to the trained model. This is insofar problematic as the designated runtime for the EHR application is a web server and accessing local files can be difficult due to user permissions and security settings. Therefore, the trained models are put into the C# project's *Resources* folder which means they are included in the compiled assembly when building the application. One can access these resources over calling `Properties.Resources.xyz` where `xyz` is the actual resource. However, this returns a byte array which is not possible to pass to the NLP library's sentence splitter and POS tagger as they expect a file path. As a solution, a wrapper class for creating and deleting temporary files is introduced (Appendix D). When objects of this class are disposed, they automatically remove the temporary files they created in the the runtime's temporary folder. By creating the temporary file in a `using` statement[10], the created object is automatically disposed off and the generated temporary file is therefore deleted after the `using` block ends.

After splitting the abstracts into sentences, they have to be transformed into the *Attribute-Relation File Format*[11] (ARFF) that can be read by *Weka* (Listing 6.6). It consists of a header containing the name of the relation (line 3), the list of attributes with their data types (lines 5-7) and a data part containing the actual ML data (line 9-13). The created ARFF relation contains three attributes: the sentence's position, the sentence text and whether or not is is a concluding sentence.

Preprocessing and Algorithm Selection

Using the created ARFF file, an input vector is created from the `sentence` attribute by using *Weka*'s `StringToWordVector`-filter. It tokenises the sentences and constructs a BoW model. The parameters of the filter include the option to lowercase all tokens, which stemmer, stop words handler and tokeniser to use and how many words to keep in the created BoW model. Altering these parameters does have a noticeable effects on the quality of the resulting ML model. Finding the best preprocessing strategies and ML algorithm for a particular ML problem is usually an iterative process. To test which strategies and algorithms were fitting best for this task, several parameter settings and two algorithms were tried. As ML algorithms, a *Naïve Bayes* and a *SVM* are selected as they usually provide good results in text mining settings.

[10] `https://msdn.microsoft.com/en-us//library/yh598w02.aspx`

[11] `http://www.cs.waikato.ac.nz/ml/weka/arff.html`

```
private static StringReader CreateArffFile(IList<Sentence>
    ↪ sentences)
{
    StringBuilder arff = new StringBuilder("@relation
        ↪ 'predict-conclusion'\n");
    //Attributes
    arff.AppendLine("@attribute pos numeric");
    arff.AppendLine("@attribute sentence string");
    arff.AppendLine("@attribute conc {False,True}");
    //Data
    arff.AppendLine("@data");
    foreach (var sentence in sentences)
    {
        arff.AppendLine(sentence.Position + ",\"" +
            ↪ sentence.Text.Replace("\"", "") + "\",False");
    }
    return new StringReader(arff.ToString());
}
```

Listing 6.6: ARFF file creation

Stemming the sentences' words showed no benefit, even seemed to worsen results and was therefore not further investigated. This is a similar observation to the one that Huang et al. (2013) made. They described that if more than 500 features were used, "there is no significant difference between the performance of classifiers that use a stemmer and those that do not." (Huang et al., 2013, p. 945). Table 6.1 shows the performance of the two selected algorithms with four different preprocessing strategies.

Feature	StopWord	Naive Bayes				SVM			
Selection	Removal	P	R	F_1	MCC	P	R	F_1	MCC
N	Y	0.928	0.931	0.929	0.716	0.933	0.920	0.924	0.728
Y	Y	0.921	0.915	0.917	0.687	0.926	0.916	0.919	0.703
N	N	0.945	0.946	0.945	0.783	0.935	0.928	0.930	0.742
Y	N	0.919	0.914	0.916	0.680	0.931	0.924	0.927	0.725

Table 6.1: Model performance to predict concluding sentences

The used data set had 19,141 instances which were split at 66% into training and test sets. The number of words to keep for the input vector was set to 2,000 words as

this is still within the dimensions a normal computer can handle. Setting the word limit to a higher value lead to slow processing and spontaneous crashes of the JVM as the computer's RAM limit was reached during training while also not providing much of a prediction benefit. It can be observed that using *Naive Bayes* algorithm without feature selection, stop word removal and stemming showed the best results with a F_1-measure of 0.945 and MCC of 0.783. Additionally, training and using a *Naive Bayes*-model is considerably faster than that of a SVM.

The resulting ML model is subsequently used during querying to predict the concluding sentences of an unstructured abstract. Similar the the NLP library's models in the previous section, the generated ML model is also stored in the project's Resources folder and accessed over the `TempFile`-class outlined in Appendix D.

6.1.5 Literature Clustering with Carrot2

There are two possible ways to integrate the clustering engine *Carrot2* into the C# EHR application. Either using it directly from code by running it in *IKVM.NET*, or by hosting a *Carrot2* document clustering server (DCS) and querying that over a REST API. Both strategies are officially supported. Table 6.2 compares the two integration possibilities. The performance tests were made on a sample collection of 70 publications and a *Windows 10* virtual machine with a 2x2.30GHz CPU and 4GB of RAM running on a SSD. Using `Console.Write()` calls, the clustering times were captured and rounded to seconds.

	Carrot2 DCS	**Carrot2 .NET API**
Scalability	High, as server can be independently scaled up or down	Low, as clustering shares resources with rest of EHR application.
Caching	Yes	No
Performance (clustering time)	Cold cache: 4 seconds, warm cache: 2 seconds	3 seconds

Table 6.2: Running Carrot2 as a server vs. running it in IKVM.NET

As the clustering quality should be the same with both solutions, it comes down to which solution offers the best clustering speed, its scalability and the convenience of using the solution. If just looking at the clustering time performance, no clear solution can favoured over the other. The DCS is slower on cold cache but outperforms the *IKVM.NET*-solution when using a warm cache. In terms of scalability, using a separate server is definitely more scalable than using the local implementation. Additionally, clustering on a server will not use resources that are needed by the actual EHR application. Also, none of the two solutions is clearly more or less convenient than the other. One could argue that when using a server, you have the overhead of a HTTP request. Nonetheless, in the end the Carrot2 DCS is used as it seems the be the cleaner solution.

Querying the Document Clustering Server

As the *Carrot2* DCS is selected as clustering solution, the documents as well as the parameters to tune the clustering algorithm have to be send over a HTTP request. Therefore, all retrieved `Literature` objects have to be serialised into one XML file that can be used to query *Carrot2*. This can be done by using *.NET*'s native XML serialisation. Before doing so, each literature's abstract is preprocessed by using regular expressions to remove numbers and the contents of parentheses. As a result, the clustering algorithms will return better results due to less noisy documents. The generated XML is added to the HTTP request as a parameter.

To tune the clustering algorithm, the specific settings are also added as HTTP request parameters. The following settings are used for clustering the `Literature` objects and were identified by trying different settings, looking at clustering results and subjectively judging the clustering quality:

- **min cluster size**: The minimum size of clusters to create. No clusters with less documents than this value are created. Value: 3
- **maxWordDf**: The maximum document frequency of a term. If a term appears in more than x% of documents, ignore it. Value: 0.7
- **scoreWeight** Balance between the cluster score and the cluster size. The resulting value will be used to order the clusters. Value: 0.7

- **factorizationfactory**: The method to create term-document matrix to generate cluster labels. Value: partial singular value decomposition factory

Additionally, the parameter “algorithm” with value “lingo” is provided to indicate the algorithm to use.

Cluster Filtering

The clustering server's JSON response is deserialised into a list of `Cluster` objects using the *Json.NET*'s[12] `JsonConvert`. Each `Cluster` object holds a list of literature Id's that occur in the cluster. The Ids are plain Integers without a reference to the actual `Literature` object. The mapping between clusters and an individual literature happens only in the client.

The generated clusters are filtered as mentioned in Section 5.2.7. After initialising the whitelist and blacklist, the EHR terms are added to the whitelist. Then, for each cluster the labels are split at non-alphanumerical characters and foreach resulting term the filtering is done. The first check is to test if the term is a whitespace or occurs in the blacklist. If that is the case, processing continues checking the next term in the cluster label. If the term occurs in the whitelist, immediately return `true`, stop processing the cluster label and keep the cluster. The third check includes querying the comparably slow *TermStore* to leverage the EHR application's ontology. If the term is found in the ontology, the cluster is also kept. As this is done after checking the black- and whitelists, the *Term Store* does not have to be queried for each term which speeds up the processing.

6.1.6 Saving Feedback

Details about the literature feedback were already discussed in Section 5.2.9. However, one implementation detail is how the feedback is saved to the *CDSS Metadata* store. All database communication in the EHR application happens over the object-relational mapping (ORM) framework *Entity Framework 6* (EF6) As the code-first principle is followed, usually no manual alterations to the database, tables or its fields are necessary. In fact, most entities that are stored in the database are auto

[12] `http://www.newtonsoft.com/json`

generated by a code generator and the database created accordingly as described by Humm et al. (2015).

The property `EhrTerms` of the `LiteratureFeedback` class (Figure 5.8) stores a list of strings. As this is supposed to be saved in the database and EF6 is not able to save collections of primitive types like `IList<string>`, another approach is needed. As seen in Listing 6.7 the collection of strings in line 2 is flattened to a string by concatenating the list by commas (line 5). As EF6 calls the get method of `EhrTermsAsStrings` upon persisting, the value of `EhrTerms` is serialised to a string, effectively only storing the `EhrTermsAsString` in the database. While creating the `LiteratureFeedback` objects from the database, the set method in line 6 is called and the `EhrTerms` list filled by splitting the saved string at the commas.

```
1 public class LiteratureFeedback : AbstractEntity {
2     public ICollection<string> EhrTerms { get; set; }
3     public string EhrTermsAsString
4     {
5         get { return EhrTerms == null ? "" : string.Join(",",
              ↪ EhrTerms); }
6         set { EhrTerms = value.Split(',').ToList(); }
7     }
8     //...
```

Listing 6.7: Saving a collection of primitive types in EF6

6.1.7 Literature Service View Component

Following the mockups from the interaction concept in Section 4.2, the decision support GUI is implemented using the JavaScript framework *AngularJS*[13] and the HTML and CSS framework *Bootstrap*[14]. The decision support service front end is separated into components according to the individual CDSS services. Views are typically implemented in HTML while the controllers use JavaScript. Both leverage *AngularJS* features like the two-way data binding between views and controllers.

After the EHR data is loaded, a request against the CDSS API controller is made containing the id of the currently opened EHR. This happens asynchronously while

[13] `https://angularjs.org/`
[14] `http://getbootstrap.com/`

the EHR is already displayed. As loading of the CDSS should not distract the user, no spinner indicates that the service is searching for literature. Instead, a single String is shown in the service panel stating "Searching Literature...". Once the data is returned by the CDSS API, the publications are parsed to highlight text, the publications and clusters are assigned to *AngularJS* variables and appropriate status messages are displayed.

Filtering Literature

On clicking a filter, the function `setActiveFilter()` is executed. It fetches the clicked filter from the list of all filters and changes its attribute `active` to `true`. Due to the two-way data binding of *AngularJS*, this is responsible for rendering the filter differently in the view. Additionally, the function extracts the filter's associated documents from it and assigns them to the variable `documentFilter`. As a result, only publications with an Id occurring in the `documentFilter` variable are displayed. This is realised by using *AngularJS*' `ng-repeat` with a filter function that utilises the `documentFilter` variable.

Search Term Highlighting

Before displaying the publications, the previously mentioned highlighting of EHR terms is executed to allow quick identification of relevant terms. This functionality is implemented completely in JavaScript. The response form the CDSS API includes the EHR terms, which are used to mark them in the publications' titles and abstracts using the function in Listing 6.8. Basically, each EHR term is searched in the provided text using a regular expression (line 6). If it is found in the text, it is surrounded with HTML `<mark>` tags that are rendered in a special way in the browser (line 8). The function `.each()` in line 5 is part of the JavaScript library *Underscore.js*[15]. The `highlight()` function is only executed once per publication.

[15] `http://underscorejs.org/`

```
function highlight(text) {
    if (_.isEmpty(vm.ehrTerms)) {
        return text;
    }
    _.each(vm.ehrTerms, function(term) {
        var regex = new RegExp(term, 'gi');
        text = text.replace(regex, function (match) {
            return '<mark>' + match + '</mark>'
        });
    });
    return text;
}
```

Listing 6.8: Function to highlight EHR terms in publications

Responsiveness Improvements

Initial user interactions in the client showed a rather disappointing responsiveness. This was due to the number of displayed publications and the complicated rendering behind it. To improve performance, the number of publications initially displayed is limited to 10 publication. By leveraging the *ngInfiniteScroll* module[16], more publications are only loaded from the controller once the user scrolls down and approaches the end of the currently loaded publication list. This leads to a significantly better interface responsiveness.

6.2 Drug Interaction Service

The drug interaction service is invoked by the REST API controller that provides a list of currently prescribed `Medication` objects as well as list of Strings with additional medications that the user entered in the GUI (Section 6.2.2). These two lists are used to search for drug interactions (Section 6.2.1).

[16] `https://sroze.github.io/ngInfiniteScroll/`

6.2.1 Drug Interaction Search

The medication names are extracted from the `Medication` objects and the list of all drug names, the currently prescribed as well as the additionally provided ones, is used to query *RxNav* in order to find the specific drug identifiers (*RxCui*). For each medication, a new request has to be made and the medication name is appended to the query as a GET parameter.

The returned *RxCui*s are extracted and used to make another *RestSharp* request to the drug interaction endpoint by concatenating all *RxCuis* with a "+" and adding the resulting string to the request as a GET parameter. All requests to *RxNav*'s REST API are made using *RestSharp*[17], a simple REST and HTTP API client for .NET. It provides the possibility to deserialise the returned JSON or XML into appropriate C# objects. For both requests there is a C# class tree describing the returned XML or JSON that is used deserialise the request's HTTP response. As those classes are too verbose and contain unnecessary information, they are translated into `DrugInteraction` objects (Figure 5.10) and then send to the client.

6.2.2 Interaction Interface

Similar to the literature service GUI component, the drug interaction interface is also implemented using JavaScript and HTML and also consists of a view and a controller. After the EHR is loaded, a request is made asynchronously to the CDSS API and the response is used to display the found drug interactions. If no interactions are found, a string with the content "No interaction found" is displayed.

Autocomplete

The user has the option to add additional drugs to the interaction checking using the EHR application's autocomplete feature, a modified version of *ngTagsInput*[18]. The search function extracts the drug names from the autocompleted input tags and queries the CDSS API by providing the patient Id and an array of Strings

[17] `http://restsharp.org/`

[18] `http://mbenford.github.io/ngTagsInput/`

containing the additional drugs. This is started automatically each time a drug is added or removed by calling the search function.

Interaction Alerts

For severe interactions an alert should be displayed. Alert handling is managed in the CDSS main GUI interface that also loads all implemented CDSS service panels. The interaction controller emits a message with the name "cdss.alert" and a data object containing the interacting drugs and a short description of the interaction. The CDSS main controller listens for these "cdss.alert" messages and manages the adding of new and deleting of old alert messages. All alerts are displayed in the CDSS main view by utilising the *AngularJS* function `ng-repeat`.

Chapter 7

Evaluation

This section evaluates the proposed CDSS by comparing concept and prototype implementation with the Requirements defined in Chapter 2. Requirement 1.2 (personalised) and 1.3 (pro-active) are obviously met since the CDSS information is displayed automatically based on the EHR currently being handled. Also, requirement 1.5 (Workflow) is met, since the CDSS panels can be separated from the EHR dialogs. Assessing the Requirement 1.1 (relevant), 1.4 (easily comprehensible) and 2.1 (usable) is less obvious and needs feedback from users. Therefore, an initial survey with 4 medical students and 1 resident physician was conducted.

7.0.1 Usability Tests and Initial Survey

Users were given the task to prepare for a multidisciplinary team (MDT) meeting using the different CDSS modules. They were observed while doing so and should communicate what they were thinking while using the system. Subsequently, they were to fill out a questionnaire to assess the usability aspects and relevance of the CDSS services implemented in the prototype as well as in the interaction concept.

The usability test with the participants revealed some weak points that were, subsequently, improved. For example, the teaser text's icon position was initially not prominent enough and was easily overlooked. Therefore, its position was moved after the literature's title and a legend was added. Another point that was introduced after conducting the usability tests was the automatic drug interaction search after the user added an additional drug in the drug interaction panel. Users were

initially confused why they had to explicitly press the button in order to search for interactions.

The initial survey indicates positive feedback. Its specific question and the answers can be seen in Appendix E. The participants could answer each question by choosing a number on a scale from 1 to 5 where 1 meant "not helpful" or "do not agree" and 5 meant the opposite. Positively mentioned were the display of the teaser text for assessing literature relevance, searchable adverse effects, the EBM recommendations service and the clinical trial locator service. Not considered as important were the news service and the display of non-severe drug interactions. Concerning the literature service, the participants agreed that the found publications would be helpful to prepare for a MDT meeting, that they were up-to-date and that the display of the teaser text helped in assessing the publications' relevance. However, the question if the literature service would be helpful in daily clinical care was answered with an average of 3.6 points. This is insofar strange as participants answered the first question ("In order to prepare for a MDT meeting, where the found publications helpful?") with 4 points in average. This result is probably caused by the lack of clinical experience of some participants. As they did not know the answer to the question, they tended to rate the question with 3 points. An option indicating "do not know" or something similar would have be better here.

Another interesting observation concerns the question if non-severe drug interactions should also be displayed in the EHR. The comparably high standard deviation indicates some disagreement among the participants. Interestingly, the group of resident physicians did not think a drug interaction service as relevant and did not want to see non-severe drug interactions. The reasons for this can only be guessed and is subject to further investigation.

7.0.2 Response Time

When opening EHR for the first time initial loading of literature data may take up to 15 seconds. However, as this happens asynchronously while the consultant is working with EHR this will not interfere with the EHR workflow. With the caching of `Literature` objects, this initial loading time is reduced to 4 seconds when opening the same EHR a second time. Finding and displaying of drug interactions is independent of that and usually takes about 2 seconds. As soon as all data is loaded

and the physician enters the CDSS, each interaction is less than 200 ms which meets Requirement 2.2 (low response time) clearly.

7.0.3 Extensibility and Maintainability

Concerning Requirement 2.3 (extensible), the component-based system architecture and the use of adapters to access information sources enables the extension of the CDSS and implement other decision support modules with moderate implementation effort. New EHR fields can be easily adopted by adding them to the *EhrMapper* and integrating them with a rule, if desired. For a new data source one would have to implement an additional adapter which results in about 100 lines of code (LOC). For a new decision support module the service on the server and the client GUI would have to be implemented. For the client this would require about 300 LOC. The server implementation depends on the module's complexity, e.g. the literature service has 2,500 LOC whereas the interaction service has about 200 LOC.

7.0.4 Literature Service Document Retrieval Evaluation

As the proposed literature service is basically a document retrieval task, it is evaluated as such. However, assessing the relevance of a publication is not an easy task and can only be done by domain experts. Even then, the votes differ depending on the individuals knowledge levels. Usually, several experts are asked to evaluate the retrieved publications and then an agreement measure like the Kappa Statistic is calculated to see how much agreement among those experts exists (Manning et al., 2008, p. 164). Unfortunately, only the feedback of one medical student was available to evaluate the retrieval quality of the literature service.

The task was to asses the relevance of the found literature for each of the three test cases and give feedback using the literature service's feedback buttons. Most of the retrieved publications were relevant to the patient as can be seen with the calculated average precision of 0.745. Leveraging this feedback, two different rankings can be evaluated. The first is the ranking prior to any feedback, henceforth called the *PubMed Ranking*. The second ranking leverages this user feedback and incorporates it into the ranking of the found literature (Section 5.2.8). This is referred to as

Feedback Ranking. Figure 7.1 shows the precision-recall curves of those two ranking variations.

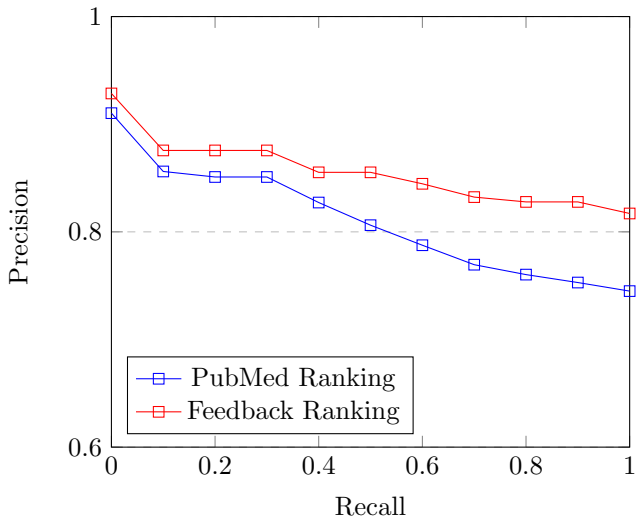

Figure 7.1: Precision-recall curves for ranked retrieval in the literature service

The graphs show the average precisions from three test cases. Although small, it can be observed that using the user feedback to modify the initial *PubMed* ranking resulted in an overall better retrieval quality. Also, the two curves seem to be similar in the first few recall levels but diverge at a recall level of around 0.5. This means that relevant documents are successfully ranked higher than non-negative ones.

Chapter 8

Related Work

The idea of CDSS is not new and there exist many services accessible over the browser and/or smartphone applications. Their scope ranges from drug information, drug interaction and diseases to guidelines, pill identification and alternative medications. Example services include *UpToDate*, *Epocrates*, *MedScape* and *First Databank*. Often, the integration of theses services into an EHR system consists of providing a standard search field that enables the users to search and visit the CDSS service's main webpage. Some EHR systems like *Athenahealth*'s EHR[1] include context-sensitive drug monographs that provide information like dosing, adverse effects and other key safety data directly in the EHR. However, there are systems that include include the patient's context in the CDSS search. The integration of *UpToDate* into various EHR systems provides such a future by displaying an info button in various EHR locations and providing an automatic search[2]. However, this function delivers standard *UpToDate* results pages and problems mentioned earlier in this work like confusingly written texts and difficulties navigating the long summaries remain (Obst et al., 2013).

Finland's Evidence based Medicine electronic Decision Support (Nyberg, 2012) is a platform-independent online service that accepts structured EHR data as input and returns clinical decision support data like links to guidelines, therapeutic suggestions, clinical reminders and alerts. These rules can be created by experts in a web-based editor and scoped per organisation or globally. It can also populate forms

[1] `http://www.athenahealth.com/enterprise/epocrates/clinical-decision-support`
[2] `http://www.uptodate.com/home/hl7`

and calculators with patient specific data. They do not include a literature search service but provide other services like the drug alerts that are similar to services presented in this work. Additional services like the knowledge assistant would be relevant in the context of this work and could later be integrated into our proposed CDSS.

Alternative approaches for the task of finding literature fitting to a patient's case has been described in different publications. Perez-Rey et al. (2012) propose a visual browser extension that allows the user to select a subset of extracted search terms from a natural language medical record. These selected terms will then be used to search PubMed. However, the selection of search terms is not automatic or pro-active as the user has to interact with the application to build the search.

Soldaini et al. (2015) propose a CDSS that tries to find fitting medical literature for medical case reports instead of EHRs. They consider the natural language case report as the query and apply query reformulation techniques like query reduction by identifying medical terms and expansion by using pseudo relevance feedback to build the search. They try to best answer the case report (ca. 60 words) by providing relevant literature. In contrast to this work they do not use PubMed as search engine but use a local search server.

To the best of our knowledge, no system integrates a literature search system into an EHR to display relevant medical literature. Similarly, we believe there is no implementation of a patient-specific search service for clinical trials.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

In this thesis, a CDSS was proposed to satisfy information needs at the point of care, especially in the context of personalised medicine. First, physician's information needs were gathered by studying scientific papers on the topic in Section 4. Based on that, a user interaction model was proposed that tries to satisfy the identified information needs by introducing serval CDSS services. After giving an overview of potential information sources that could be used for the CDSS, the CDSS software architecture was described in more detail with a focus on the literature service and the drug interaction service in Chapter 5. The literature service leverages a rule-based in combination with a machine learning approach to generate *PubMed* queries. Found literature is processed by preparing a teaser text as well as filters to quickly navigate the literature set. Special focus was also on how to effectively display the found data to the user. Another service architecture was described with the drug interaction service that leverages *RxNav* as data source and allows users to manually add additional drugs to search for interactions.

Those two services' implementation was detailed in Section 6 with a focus on language-specific details and encountered difficulties while implementing. In Chapter 7 the proposed CDSS was finally evaluated against the initially described requirements. It could be shown that all of the requirements were met. However, the evaluation only featured 5 participants of which 4 were medical students. A study with

medical experts might yield different results and is highly recommended by the author of this study.

Particular focus has been on the usability of the CDSS allowing consultants to quickly and intuitively gather relevant information with minimal system interaction. This includes but is not limited to the automatic extraction of a literature abstract's conclusion using ML, the marking of EHR terms in found literature titles and abstracts, the creation of clusters to filter the literature set and usability tests with users to see where confusion might occur.

Personalised medicine in general offers great promises for treatment response prediction and physician decision making and can therefore significantly improve patients' health and safety. Numerous medical information sources are already available which can be utilised, and they are growing rapidly. However, it seems that those sources are not yet fully integrated and used in day-to-day clinical practise although the integration of decision support service into clinical software yields many benefits. With the concept and a prototypical implementation of a CDSS for personalised medicine presented in this paper, a contribution towards this direction is made. May this work eventually help consultants improve patient care.

9.2 Future Work

It is planned to integrate the CDSS presented into a commercial EHR application suite for melanoma treatment. Towards this end, future work is required. Additional information services as presented in the interaction concept (Section 4.2) need to be implemented. A comprehensive analysis of the CDSS with consultants in the field needs to take place resulting in potential improvements of the concept and the implementation. Also, trial phase with real patient data is necessary to extend the data base for machine learning and get realistic user feedback.

9.2.1 Additional CDSS Services

New CDSS services include the display of context sensitive medical calculators. An example for this would be to calculate right drug dosing for elderly patients with impaired renal function using the glomeral filtration rate (Rahmner et al., 2012).

In order to provide the right calculators at the right time, close work with experts in the field is necessary. Another point Rahmner et al. (2012) mentioned were patients' hypersensitivities. Study participants said that if a sensitivity is detected, the system should prohibit prescription. This is the only time when the system actively interacts instead of just providing guidance and decision support. In order to provide this functionality, the EHR application would need information on a patient's hypersensitivities in the form of a new EHR field. Additionally, it would make sense to store this kind of information not in a system in one health organisation, but rather encourage the use of interpretability and information exchange between systems of different health institutions. This would meet the definition of an EHR mentioned in Section 3.2.

9.2.2 Standards and Service Architecture

Apart from the CDSS services introduced in the Interaction Concept, CDSS modules from other clinical decision support providers like *UpToDate.com* or *EBMeDS* could be integrated into this system. To do so, it would be beneficial to follow specific clinical decision support standards like the *HL7 Decision Support Service* standard[1].

Additionally, as current developments in the domain of CDSS architecture tend to adopt a service architecture approach, this could be adopted for the proposed CDSS. In a service architecture, the clinical information system information is clearly separated from the CDSS and connected over standardised service-based interfaces (Sanchez, 2014, p. 45). While the CDSS proposed in this work already tries to separate the EHR from the CDSS by using the *EhrMapper*, efforts could be increased to create for example a web-based CDSS that can be used from a variety of systems. This approach is similar to the one *EBMeDS* is employing and allows to support a wide range of clinical systems.

9.2.3 Extending the Literature Service

Several improvements and extension can be made to the literature service. First, most EHR fields should be used and mapped in the *EhrMapper*. Then, additional rules incorporating these fields would be created in cooperation with medical experts.

[1] `http://www.hl7.org/implement/standards/product_brief.cfm?product_id=12`

As rule queries are created using expression trees that are built from a composite pattern, this rule creation could be made more dynamic. There is even the possibility to allow users to define rules that could be used in conjunction with the already existing ones.

Learning to Rank

In the proposed literature search service ranking of results is done by using vector-space-based features like the inverse document frequency in combination with active and passive user feedback. Both of those relevance assessment measures are then combined by applying weights that have to be manually adjusted. The actual order of publications therefore depends on those weights and needs constant intervention in the source code of the literature search, at least until good weights are found. A more desirable approach would be to train a ML algorithm to learn those weights or to rank the found literature according to a learned model. That technique is used by many popular search engines and referred to as a *learning to rank* task. Other features describing the literature could be included in the ranking, e.g. the impact factor of the journal or the *Altmetric*[2] score indicating how many people are talking about a publication.

Personal Feedback and Literature Search

Once the EHR application features a user management functionality, the feedback given by a specific user could be stored and displayed whenever he/she revisits the specific literature. Additionally, a more personalised literature search strategy would be possible as information demand probably differs from person to person depending on their level of experience and knowledge. The feedback of a user could therefore be used to create a personalised search.

Machine Learning Training

Training of the SVM is currently initiated over the test project. However, once the EHR application features a user management and different user groups, an

[2] `https://www.altmetric.com/`

administrator could be given the possibility to initiate SVM training over the GUI. Another approach is to automatically trigger SVM training once a certain amount of feedback is gathered. As one would have to make sure that training happened not during the application's peak usage times, a combination of both approaches might be beneficial where an administrator gets a notification and can decide when to do another ML run.

Better Term Extraction

The current solution for extraction ontology terms from literature uses the EHR applications *Term Store*. However, that solution is rather slow as many possible terms are created and searched for in the *Term Store*. This has two disadvantages. First, this leads to a lengthy process during the ML training for query term prediction. Second, while the ML query prediction is trained, the *Term Store* might not respond as fast as desired to autosuggest calls from the actual EHR application. As these should never be slowed, another approach to identify terms in a literature is desirable. Therefore, Named Entity Recognition tools like *MetaMap*[3] could be integrated to identify medical concepts in a text. Another possibility is to use solely the literature's MeSH terms instead of extracting terms from title and abstract. For publications that are not yet indexed with MeSH terms, a ML approach could be applied to predict the right terms.

9.2.4 Drug Interaction Data Source

As already stated earlier, the data source for the drug interaction service is not usable in a production systems as no information about the interaction severity is provided. Further work has to be done in order to find and integrate an appropriate data source. This will probably be a commercial solution.

[3] `https://metamap.nlm.nih.gov/MetaMapLite.shtml`

Appendices

Appendix A

Literature Service Data Sources

This table lists the identified data sources for the literature service. "Public & free" access means only the access to the search interface and abstracts. Full publications are nonetheless sometimes only available after payment. Also, the volume and availability of an API could not be identified for all databases.

Name	**Description**	**API**	**Access**	**Volume**
Cochrane Library	Collection of health-related databases. Its core is *Cochrane Reviews*, a database of systematic reviews and meta-analyses.	?	subscription	?
Google Scholar	Search engine for scientific publications of all fields. Automatically crawls many journals.	no	public & free	estimated at 160 million articles
Ovid	Science search platform that includes many databases, including MEDLINE.	?	subscription	?

PubMed	Search engine mainly accessing MEDLINE database and focused on health topics. Query expansion by using MeSH ontology.	yes	public & free	> 24.6 million records, about 500,000 new records each year
ScienceDirect	Website with access to large database of scientific publications from many fields.	yes	free (abstracts), subscription (full-text)	12 million records from 3,500 journals and 34,000 eBooks
Scopus	Database with abstracts and citations from many academic journals and many scientific fields, not focused on health topics.	yes	paid subscription	~60 million records, >21,500 peer-reviewed journals
Springer API	Access to all Springer published journals, also includes BioMedCentral open-access publications.	yes	partly free, partly subscription	~2,000 journals and >6,500 books per year, access to >10 million online documents

Appendix B

EBM Recommendation Sources

Name	Description	API	Access	Volume
BMJ Best Practice	Evidence-based information to offer step-by-step guidance on diagnosis, prognosis, treatment and prevention.	yes	subscription	?
DynaMedPlus	Evidence-based clinical overviews and recommendations. Content updated daily. Also offers calculators, decision trees and unit and dose converters.	yes	subscription	> 3,200 topics and > 500 journals
EBMeDS	Platform-Independent web service CDSS with EBM module	yes	commercial	
Essential Evidence	POC system with topics, guidelines, abstracts, and summaries of most common clinical cases. Also links to other resources like Cochrane Library and Evidence-Based Medicine Guidelines.	?	subscription	> 13,000 topics, guidelines, abstracts & summaries
Medscape / eMedicine	Largest clinical knowledge base available freely. Articles updated yearly. Also available as mobile application.	no	free, registration required	~6,800 articles

Physician Data Query	Cancer database from the U.S. *National Cancer Institute*. Contains peer-reviewed information on cancer treatment in the form of summaries for patients and professionals.	no	public	Only cancer domain
UpToDate	Popular evidence-based POC tool for a wide range of disciplines but targeted on internal medicine. Extensive peer-review process to ensure accurate and precise recommendations.	yes	subscription, some articles free	~8,500 topics

Appendix C

Drug Information Data Sources

The following table lists identified drug resources. With the column title *Drug Information*, all drug related material like dosing, adverse effects, administration information, pill images and patient information is understood. As there are data sources that specifically target drug interactions, adverse effects or drug announcements and recalls, those three are stated separately. The column "Access" describes how the content can be accessed. "Public & free" means no cost occur and there is no registration needed. Some services are free but require a registration. Subscriptions mean paid access to the sources. The term "commercial" describes other payment methods than subscription.

Name	Description	API	Access	Drug Information	Drug Interactions	Adverse Events	Drug Announcements/Recalls
DailyMed	Website by U.S. National Library of Medicine (NLM), provides high quality and up-to-date drug labels. Updated daily by FDA. Documents use structured XML format.	yes	public & free	✓	✓	✓	
DrugBank	Database with pharmacological drug information on drugs and their targets. Longer update periods. Links to DrugBank for nearly all drugs on Wikipedia. Drug interactions feature no information on their severity.	yes	public & free		✓		
Drugs.com	Website with drug information, pill identification and drug interaction checker for patients and for health professionals. Also provides data on modified drug labels. Prohibited to incorporate into any kind of IR system.	no	public & free	✓	✓	✓	
Electronic Medicines Compendium	Information on drugs licensed for use in the UK. Contains Summaries of Product Characteristics and Patient Information Leaflets	no	public & free	✓	✓	✓	
Epocrates	Point-of-care medical information about drugs, diseases and diagnostic tools over website or mobile app. Also features news feed of product announcements and medical news.	no	partly free / subscription needed.	✓	✓	✓	✓

MedlinePlus Connect	Service by NLM, provides unstructured natural language drug information/labelling and health topic overviews	yes**	public & free	✓*		✓*	
Medscape	Many clinical information resources available over website or mobile app. Articles updated yearly.	no	free, registration required	✓	✓	✓	
OpenFDA API	Public API on reported Adverse Effects, drug labelling and drug recall reports. Data consists of individual reports and has to be aggregated in order to use it.	yes	public & free			✓	✓
ResearchAE	Adverse effects experimental research application based on OpenFDA data. Not to be used in clinical settings.	no	public & free			✓	✓
RxNav	Provides access to different drug resources like *RxNorm*, *NDF-RT* and *DrugBank*. Drug normalisation over different codes and systems by using *RxNorm*, drug interactions from *DrugBank*.	yes	public & free		✓		
SIDER	Aggregated data on side effects for drug target prediction from publicly available sources. Infrequent updates. Download of dataset possible.	no	public & free			✓	
Wolters Kluwer Clinical Drug Information	Commercial drug information APIs including interaction, adverse effects, indications and mapping to RxNorm.	yes	commercial	✓	✓	✓	

* = Unstructured data/natural language text; ** = Only to search for links to the articles

Appendix D

Temporary File Wrapper

The following code represents the wrapper for temporary files. The class implements the `IDisposable` interface and deletes the generated temporary files when the object is either disposed or garbage collected.

```
1  public class TempFile : IDisposable
2  {
3      private string _path;
4
5      public TempFile(byte[] bytes) : this()
6      {
7          File.WriteAllBytes(_path, bytes);
8      }
9
10     public TempFile() :
           ↪ this(System.IO.Path.GetTempFileName()) { }
11
12     public TempFile(string path)
13     {
14         if (string.IsNullOrEmpty(path)) throw new
               ↪ ArgumentNullException("_path");
15         this._path = path;
16     }
17
18     public string Path
19     {
20         get
21         {
22             if (_path == null) throw new
                   ↪ ObjectDisposedException(GetType().Name);
```

```
            return _path;
        }
    }
    ~TempFile() { Dispose(false); }
    public void Dispose() { Dispose(true); }
    private void Dispose(bool disposing)
    {
        if (disposing)
        {
            GC.SuppressFinalize(this);
        }
        if (_path != null)
        {
            try { File.Delete(_path); }
            catch { } // best effort
            _path = null;
        }
    }
}
```

Appendix E

Evaluation Survey

The evaluation user survey was conducted with 4 medical students and 1 resident physician. They were to answer the following questions by choosing a number from 1 to 5, with 1 being the lowest answer option and 5 the highest.

Question	Average	Median	Std. Dev.
Literature Service			
In order to prepare for a medical discussion meeting, were the found publications helpful?	4	4	0.707
Were the found publications up-to-date?	4.6	5	0.547
Did the found publications match the patient at hand?	3.2	3	0.447
Did the filters help you in finding publications?	3.8	3	1.095
How satisfied were you with the quality of the filters? Did they accurately describe the displayed publications?	4	4	0.816
How useful did you find the display of the abstract's conclusion?	4.6	5	0.894
How would you rate the user-friendliness of the system?	4.25	4	0.5

Could you imagine the literature service as helpful in daily clinical care?	3.6	4	0.547
Drug Interaction Service			
Relevancy of such a service integrated into the EHR.	4.2	5	1.304
Non-severe interactions should also be displayed in the EHR (i.e. in a separate panel for non-severe interactions)	3.4	4	1.817
Drug Information Service			
By looking at the layout, I think I would find all information I would need to prescribe a specific drug and inform the patient.	4.4	4	0.548
Such a service would be useful in an electronic health record.	4.6	5	0.548
A service that allows searching all adverse effects of a patient's drugs could help in clinical situations.	4.8	5	0.447
Other			
Consider the field of melanoma treatment. Would a service that searches for relevant nearby clinical trials be helpful?	4.2	4	0.447
A service that provides evidence-based guidelines and reviews on treatment and diagnosis could be helpful.	4.4	5	0.894
A news service that displays latest news related to the patient could be useful.	3.6	4	1.140

Bibliography

Academy of Medical Science (2015). *Stratified, personalised or P4 medicine: a new direction for placing the patient at the centre of healthcare and health education.* Tech. rep. URL: https://www.acmedsci.ac.uk/viewFile/564091e072d41.pdf.

Agichtein, E., E. Brill, and S. Dumais (2006). "Improving web search ranking by incorporating user behavior information". In: *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, pp. 19–26. ISBN: 1595933697.

Baldi, P., S. Brunak, Y. Chauvin, C. A. Andersen, and H. Nielsen (2000). "Assessing the accuracy of prediction algorithms for classification: an overview". In: *Bioinformatics* 16.5, pp. 412–24. URL: https://www.ncbi.nlm.nih.gov/pubmed/10871264.

Banzi, R., M. Cinquini, A. Liberati, I. Moschetti, V. Pecoraro, L. Tagliabue, and L. Moja (2011). "Speed of updating online evidence based point of care summaries: prospective cohort analysis". In: *BMJ* 343, p. d5856. DOI: 10.1136/bmj.d5856.

Beez, U. (2015). "Terminology-Based Retrieval of Medical Publications". Thesis. University of Applied Science Darmstadt.

Beez, U., B. G. Humm, and P. Walsh (2015). "Semantic AutoSuggest for Electronic Health Records". In: *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 760–765. DOI: 10.1109/csci.2015.85.

Berner, E. S. (2009). *Clinical Decision Support Systems: State of the Art.* Tech. rep. Agency for Healthcare Research, Quality (U.S. Department of Health, and

Human Services). URL: https://healthit.ahrq.gov/sites/default/files/docs/page/09-0069-EF_1.pdf.

Berner, E. S. and T. J. La Lande (2007). "Overview of Clinical Decision Support Systems". In: *Clinical Decision Support Systems: Theory and Practice*. Ed. by E. S. Berner. New York, NY: Springer New York, pp. 3–22. ISBN: 978-0-387-38319-4. DOI: 10.1007/978-0-387-38319-4_1.

Bohm, R., L. von Hehn, T. Herdegen, H. J. Klein, O. Bruhn, H. Petri, and J. Hocker (2016). "OpenVigil FDA - Inspection of U.S. American Adverse Drug Events Pharmacovigilance Data and Novel Clinical Applications". In: *PLoS One* 11.6, e0157753. ISSN: 1932-6203 (Electronic) 1932-6203 (Linking). DOI: 10.1371/journal.pone.0157753.

Bundesministerium für Gesundheit (2016). *Medikationsplan unterstützt Patienten, Ärzte und Apotheker*. URL: http://www.bmg.bund.de/ministerium/meldungen/2016/medikationsplan.html (visited on 04/10/2016).

Charniak, E. (1997). "Statistical techniques for natural language parsing". In: *AI magazine* 18.4, p. 33.

Clarke, M. A., J. L. Belden, R. J. Koopman, L. M. Steege, J. L. Moore, S. M. Canfield, and M. S. Kim (2013). "Information needs and information-seeking behaviour analysis of primary care physicians and nurses: a literature review". In: *Health Info Libr J* 30.3, pp. 178–90. ISSN: 1471-1842 (Electronic) 1471-1834 (Linking). DOI: 10.1111/hir.12036.

Coalition, P. M. (2014). *The Case for Personalized Medicine - 4th Edition*. Tech. rep. URL: http://www.personalizedmedicinecoalition.org/Userfiles/PMC-Corporate/file/pmc_the_case_for_personalized_medicine.pdf.

Ebell, M. H., R. Cervero, and E. Joaquin (2011). "Questions asked by physicians as the basis for continuing education needs assessment". In: *J Contin Educ Health Prof* 31.1, pp. 3–14. ISSN: 1554-558X (Electronic) 0894-1912 (Linking). DOI: 10.1002/chp.20095.

Falagas, M. E., E. I. Pitsouni, G. A. Malietzis, and G. Pappas (2008). "Comparison of PubMed, Scopus, Web of Science, and Google Scholar: strengths and weaknesses". In: *FASEB J* 22.2, pp. 338–42. ISSN: 1530-6860 (Electronic) 0892-6638 (Linking). DOI: 10.1096/fj.07-9492LSF.

Garret, P. and S. Joshua (2011). *EMR vs EHR – What is the Difference?* URL: ttps://www.healthit.gov/buzz-blog/electronic-health-and-medical-records/emr-vs-ehr-difference/ (visited on 05/10/2016).

Huang, K.-C., I. J. Chiang, F. Xiao, C.-C. Liao, C. C.-H. Liu, and J.-M. Wong (2013). "PICO element detection in medical text without metadata: Are first sentences enough?" In: *Journal of Biomedical Informatics* 46.5, pp. 940–946. ISSN: 1532-0464. DOI: http://dx.doi.org/10.1016/j.jbi.2013.07.009. URL: http://www.sciencedirect.com/science/article/pii/S153204641300110X.

Humm, B. G. and P. Walsh (2015). "Flexible yet Efficient Management of Electronic Health Records". In: *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 771–775. DOI: 10.1109/csci.2015.84.

Hung, B. T., N. P. Long, P. Hung le, N. T. Luan, N. H. Anh, T. D. Nghi, M. V. Hieu, N. T. Trang, H. F. Rafidinarivo, N. K. Anh, D. Hawkes, N. T. Huy, and K. Hirayama (2015). "Research trends in evidence-based medicine: a joinpoint regression analysis of more than 50 years of publication data". In: *PLoS One* 10.4, e0121054. ISSN: 1932-6203 (Electronic) 1932-6203 (Linking). DOI: 10.1371/journal.pone.0121054.

ISO (2005). *Health informatics — Electronic health record — Definition, scope and context*. ISO ISO/TR 20514:2005. Geneva, Switzerland: International Organization for Standardization.

Karthigeyan, L., P. Murugesan, and S. P. Natarajan (2014). *Clinical Decision Support Systems & Tools*. Tech. rep. Computer Sciences Corporation. URL: http://assets1.csc.com/innovation/downloads/Clinical_Decision_Support_Systems.pdf.

Kwag, K. H., M. Gonzalez-Lorenzo, R. Banzi, S. Bonovas, and L. Moja (2016). "Providing Doctors With High-Quality Information: An Updated Evaluation of Web-Based Point-of-Care Information Summaries". In: *J Med Internet Res* 18.1, e15. ISSN: 1438-8871 (Electronic) 1438-8871 (Linking). DOI: 10.2196/jmir.5234.

Lyman, J. A., W. F. Cohn, M. Bloomrosen, and D. E. Detmer (2010). "Clinical decision support: progress and opportunities". In: *J Am Med Inform Assoc* 17.5, pp. 487–92. ISSN: 1527-974X (Electronic) 1067-5027 (Linking). DOI: 10.1136/jamia.2010.005561.

Maggio, L. A., R. M. Steinberg, L. Moorhead, B. O'Brien, and J. Willinsky (2013). "Access of primary and secondary literature by health personnel in an academic health center: implications for open access". In: *J Med Libr Assoc* 101.3, pp. 205–12. ISSN: 1558-9439 (Electronic) 1536-5050 (Linking). DOI: 10.3163/1536-5050.101.3.010.

Maggio, L. A., O. T. Cate, L. L. Moorhead, F. van Stiphout, B. M. Kramer, E. Ter Braak, K. Posley, D. Irby, and B. C. O'Brien (2014). "Characterizing physicians' information needs at the point of care". In: *Perspect Med Educ* 3.5, pp. 332–42. ISSN: 2212-2761 (Print) 2212-2761 (Linking). DOI: 10.1007/s40037-014-0118-z.

Manning, C., P. Raghavan, and H. Schütze (2008). *Introduction to Information Retrieval*. Cambridge University Press, p. 496. ISBN: 0521865719, 9780521865715.

Marchant, G. E. and R. A. Lindor (2013). "Personalized medicine and genetic malpractice". In: *Genet Med* 15.12, pp. 921–2. ISSN: 1530-0366 (Electronic) 1098-3600 (Linking). DOI: 10.1038/gim.2013.142.

Matthews, B. W. (1975). "Comparison of the predicted and observed secondary structure of T4 phage lysozyme". In: *Biochim Biophys Acta* 405.2, pp. 442–51. URL: https://www.ncbi.nlm.nih.gov/pubmed/1180967.

Mohri, M., A. Rostamizadeh, and A. Talwalkar (2012). *Foundations of Machine Learning*. The MIT Press, p. 480. ISBN: 026201825X, 9780262018258.

National Cancer Institute (2016). *Treatment Option Overview for Melanoma*. URL: http://www.cancer.gov/types/skin/hp/melanoma-treatment-pdq#section/_885 (visited on 06/09/2016).

NCBI (2016). *PubMed Help [Internet]*. URL: http://www.ncbi.nlm.nih.gov/books/NBK3827/ (visited on 24/09/2016).

NLM Tech Bull (2013). *PubMed Relevance Sort*. URL: https://www.nlm.nih.gov/pubs/techbull/so13/so13_pm_relevance.html (visited on 26/09/2016).

Nyberg, P. (2012). *EBMeDS Clinical Decision Support. EBMeDS White Paper*. Tech. rep. Duodecim Medical Publications Ltd. URL: http://www.ebmeds.org/www/EBMeDS%20White%20Paper.pdf.

Obst, O., C. Hofmann, H. Knüttel, and P. Zöller (2013). ""Ask a question, get an answer, continue your work!" – Survey on the use of UpToDate at the universities of Freiburg, Leipzig, Münster and Regensburg". In: *GMS Med Bibl Inf* 13.3, p. 26. DOI: 10.3205/mbi000290.

Osiński, S. and D. Weiss (2016). *Carrot2 - User and Developer Manual for version 3.14.0-SNAPSHOT*. URL: http://download.carrot2.org/head/manual/index.html#table.lingo-stc-characteristics (visited on 01/10/2016).

Osiński, S., J. Stefanowski, and D. Weiss (2004). "Lingo: Search results clustering algorithm based on singular value decomposition". In: *Intelligent information processing and web mining*. Springer, pp. 359–368.

Perez-Rey, D., A. Jimenez-Castellanos, M. Garcia-Remesal, J. Crespo, and V. Maojo (2012). "CDAPubMed: a browser extension to retrieve EHR-based biomedical literature". In: *BMC Med Inform Decis Mak* 12, p. 29. DOI: 10.1186/1472-6947-12-29. URL: https://www.ncbi.nlm.nih.gov/pubmed/22480327.

Peters, L. B., N. Bahr, and O. Bodenreider (2015). "Evaluating drug-drug interaction information in NDF-RT and DrugBank". In: *J Biomed Semantics* 6, p. 19. ISSN: 2041-1480 (Electronic). DOI: 10.1186/s13326-015-0018-0.

Power, J. D. (2008). "Decision Support Systems: A Historical Overview". In: *Handbook on Decision Support Systems 1: Basic Themes*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 121–140. ISBN: 978-3-540-48713-5. DOI: 10.1007/978-3-540-48713-5_7.

Rahmner, P. B., B. Eiermann, S. Korkmaz, L. L. Gustafsson, M. Gruven, S. Maxwell, H. G. Eichle, and A. Veg (2012). "Physicians' reported needs of drug information at point of care in Sweden". In: *Br J Clin Pharmacol* 73.1, pp. 115–25. ISSN: 1365-2125 (Electronic) 0306-5251 (Linking). DOI: 10.1111/j.1365-2125.2011.04058.x.

Rechenthin, M. D. (2014). "Machine-learning classification techniques for the analysis and prediction of high-frequency stock direction". Doctoral dissertation. University of Iowa. URL: http://ir.uiowa.edu/cgi/viewcontent.cgi?article=5248&context=etd.

Redekop, W. K. and D. Mladsi (2013). "The faces of personalized medicine: a framework for understanding its meaning and scope". In: *Value Health* 16.6 Suppl, S4–9. DOI: 10.1016/j.jval.2013.06.005.

Rishel, W., T. J. Handler, and J. Edwards (2005). *A Clear Definition of the Electronic Health Record*. Tech. rep. Gartner, Inc. URL: https://www.gartner.com/doc/485998/clear-definition-electronic-health-record.

Sammut, C. and G. I. Webb (2011). *Encyclopedia of Machine Learning*. Springer Publishing Company, Incorporated, p. 1058. ISBN: 0387307680, 9780387307688.

Sanchez, E. (2014). "Semantically Steered Clinical Decision Support Systems". Doctoral dissertation. University of the Basque Country. URL: http://www.ehu.eus/ccwintco/uploads/7/7c/Eider_Sanchez_Tesis_20140220_Printed.pdf.

ScienceDaily (2016). *Terms and Conditions of Use*. URL: https://www.sciencedaily.com/terms.htm (visited on 10/09/2016).

Smith, R. (1996). "What clinical information do doctors need?" In: *BMJ : British Medical Journal* 313.7064, pp. 1062–1068. ISSN: 0959-8138 1468-5833. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2352351/.

Soldaini, L., A. Cohan, A. Yates, N. Goharian, and O. Frieder (2015). "Retrieving Medical Literature for Clinical Decision Support". In: *Advances in Information Retrieval: 37th European Conference on IR Research, ECIR 2015, Vienna, Austria, March 29 - April 2, 2015. Proceedings*. Ed. by A. Hanbury, G. Kazai, A. Rauber, and N. Fuhr. Cham: Springer International Publishing, pp. 538–549. ISBN: 978-3-319-16354-3. DOI: 10.1007/978-3-319-16354-3_59.

Sugiyama, K. (2004). "Studies on Improving Retrieval Accuracy in Web Information Retrieval". Doctoral dissertation. Nara Institute of Science and Technology. URL: https://www.comp.nus.edu.sg/~sugiyama/papers/SugiyamaFinalDissertation.pdf.

Wang, L. M., M. Wong, J. M. Lightwood, and C. M. Cheng (2010). "Black box warning contraindicated comedications: concordance among three major drug interaction screening programs". In: *Ann Pharmacother* 44.1, pp. 28–34. ISSN: 1542-6270 (Electronic) 1060-0280 (Linking). DOI: 10.1345/aph.1M475.

Webb, G. I. (2010). "Naïve Bayes". In: *Encyclopedia of Machine Learning*. Ed. by C. Sammut and G. I. Webb. Boston, MA: Springer US, pp. 713–714. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_576.

Wright, A. and D. F. Sittig (2008). "A four-phase model of the evolution of clinical decision support architectures". In: *Int J Med Inform* 77.10, pp. 641–9. DOI: 10.1016/j.ijmedinf.2008.01.004.

Zhang, X. (2010). "Support Vector Machines". In: *Encyclopedia of Machine Learning*. Ed. by C. Sammut and G. I. Webb. Boston, MA: Springer US, pp. 941–946. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_804. URL: http://dx.doi.org/10.1007/978-0-387-30164-8_804.