Grant Agreement Number: 257528

KHRESMOI

www.khresmoi.eu

D3.3: Report on query specification, result presentation and personalization

Deliverable number	*D3.3*
Dissemination level	*Public*
Delivery data	*28.2.2013*
Status	*Final*
Authors	*Sebastian Dungs, Thomas Beckers, Matthias Jordan, Sascha Kriewel, Nolan Lawson, Ivan Martinez, Andreas Tacke, Miguel Ángel Tinte*

This project is supported by the European Commission under the Information and Communication Technologies (ICT) Theme of the 7th Framework Programme for Research and Technological Development.

Executive Summary

This deliverable describes the query support, result presentation and personalization components of the Khresmoi user interface. The frontend framework and general design principles are based on the *ezDL* software. The internal processing of queries is described including query parsing and syntactic conversion. Several different representations of result items of different media types have been developed and are described. In addition, user interface elements that allow interaction with result items on different levels (result list, previews, extracted information) are presented. The user interface provides customization options and includes a personal user profile built by the system during the course of a search – which can be used to aid users in the query formulation process.

Table of Contents

List of Figures

Abbreviations

ANTLR	ANother Tool for Language Recognition (a parser generator)
API	Application Programming Interface
ASK	Anomalous State of Knowledge
BibTeX	Bibliography software (and file format) for LaTeX
CBR	Case-based reasoning
DL	Digital Library
DNF	Disjunctive Normal Form
ezDL	easy Access to Digital Libraries
GWT	Google Web Toolkit
HTTP	Hypertext Transfer Protocol
IR	Information Retrieval
KMI	Khresmoi Mimír Interface
OSGi	Open Services Gateway initiative (service delivery platform for Java)
OWL	Web Ontology Language
OWLIM	OWL semantic repository
pLib	Personal Library (a Khresmoi tool for storing and sharing search results)
RDF	Resource Description Framework
RDFS	RDF Schema (vocabulary description language for RDF)
RIS	Reference Information Systems (file format for bibliographies)
SAIL	Storage and Inference Layer
SPARQL	Query Language for RDF
UI	User Interface
URI	Uniform Resource Identifier
XML	Extensible Markup Language

1 Introduction

This deliverable describes the outcomes of project task 3.2. The results cover three aspects of the Khresmoi user interface: query specification, result presentation and interface personalization.

The term query specification describes the entirety of the users' process of entering and issuing a query to the system. This process may also include steps like disambiguation, translation and spelling correction – as well as search planning, query reformulation, and other search continuation steps. User tests and studies (e.g. [30]) have shown that users choose various strategies for query formulation which each demand different ways of supporting them. Often, poor query formulations can lead to suboptimal result sets. Furthermore, all users will benefit from system support offered during this step in the search process. Therefore, most of the work done in task 3.2. focussed on improving the system's ability of supporting query formulations.

The aspect of result presentation covers all user interface elements that display and interact with the search result set. In particular for large result sets it is important that users can quickly access a document's relevancy for their current task. Several interface features have been developed or integrated in this task, including different visual representations of single result items as well as different result set layouts. Furthermore, functionality was provided to allow users interaction with a result set by using filters or faceted navigation.

The interface is also highly flexible and can easily be adapted to the users' or a hosting organization's needs. The system offers a small set of pre-defined perspectives, suitable for the most common use cases. In addition, users can chose to rearrange tools, to resize them, or to add or remove tools from a perspective. All changes can be saved on the user's local machine and persisted between search sessions. The system can also use a user's current location as well as the preferred language to customize the search results. Information about registered users is stored in a user profile which can be manually modified. Based on monitored behaviour and the current search situation the system can provide advice for furthering a current or for improving future searches.

Chapter 2 will cover the technical fundamentals of the Khresmoi user interface as well as general design guidelines that were used. In Chapter 3 the query specification as well as the various forms of search support through suggestions are described. All aspects concerning result presentation are detailed in Chapter 4, while personalization and task-based customization is covered in Chapter 5. The last Chapter provides a summary of the findings and gives an outlook about future developments.

2 The *ezDL* frontend

The Khresmoi search user interface is based on the frontend of *ezDL*, an open-source search system [5]. The *ezDL* framework can be used to build highly interactive search systems and follows the ideas developed within the Daffodil project from 2000 to 2009 [10, 16, 21].

The architecture of *ezDL* is divided into a backend, which provides the core infrastructure through a number of software agents or services, and multiple frontends. Among these backend agents are the connectors to remote search systems and support services, user authentification and authorization, logging, a repository of known documents and a personal library in which registered users can keep a personal portfolio of found documents.

Two full-featured implementations of the frontend are available: a desktop client written in Java Swing (in the following document referred to as Swing user interface) and a web application using Google Web Toolkit (referred to as GWT user interface). Both connect to the same backend services, provide many of the same tools and follow the same design principles. This showcases how the common backend services can be accessed through multiple channels. A client for Android devices is under development.

One of the main features of the *ezDL* frontend is its configurability through the use of perspectives. Perspectives are task-based combinations of search and support tools that can be pre-configured or arranged (and saved) by the user. Perspectives may range from very simple (containing only one or two tool views) to rather complex (for expert users involved in difficult information tasks). Figure 1 shows the simple home perspective for the GWT user interface after performing a search, while Fig. 2 shows a comprehensive perspective for a "power" user containing many different tools.

2.1 Architecture

The basic component of the *ezDL* user interface is called a *tool* [5]. Tools are bundles of related functionalities which users can interact with through one or more *tool views* – interface components in the form of a windowlet. Within the user interface a specific combination and arrangement of available tools and their views on the desktop is called a *perspective*, which can be modified and/or created by users. The following tools are available for the Khresmoi search user interface (see also Fig. 3):

- The *Search Tool* is the central tool of the system and provides the most views. It offers a number of different search forms for various purposes and tasks, as well as views to present the results in list or grid form. A view for source selection allows users to switch specific information sources or media types on and off.

  The result views provide functionalities for sorting, filtering and grouping result items, and allow items to be opened in an internal preview tool (the Detail Viewer) or in a browser. Items can also be exported to a number of bibliographic formats (e.g., RIS or BibTeX). Frequent terms, authors or other features from the result set can be extracted and visualized as a list, a bar chart or a term cloud. Grouping allows for facetted browsing, and both grouping criteria and extraction strategies are encapsulated to make them easy to extend or configure. Result renderers can likewise be configured by the user or administrator of the system.

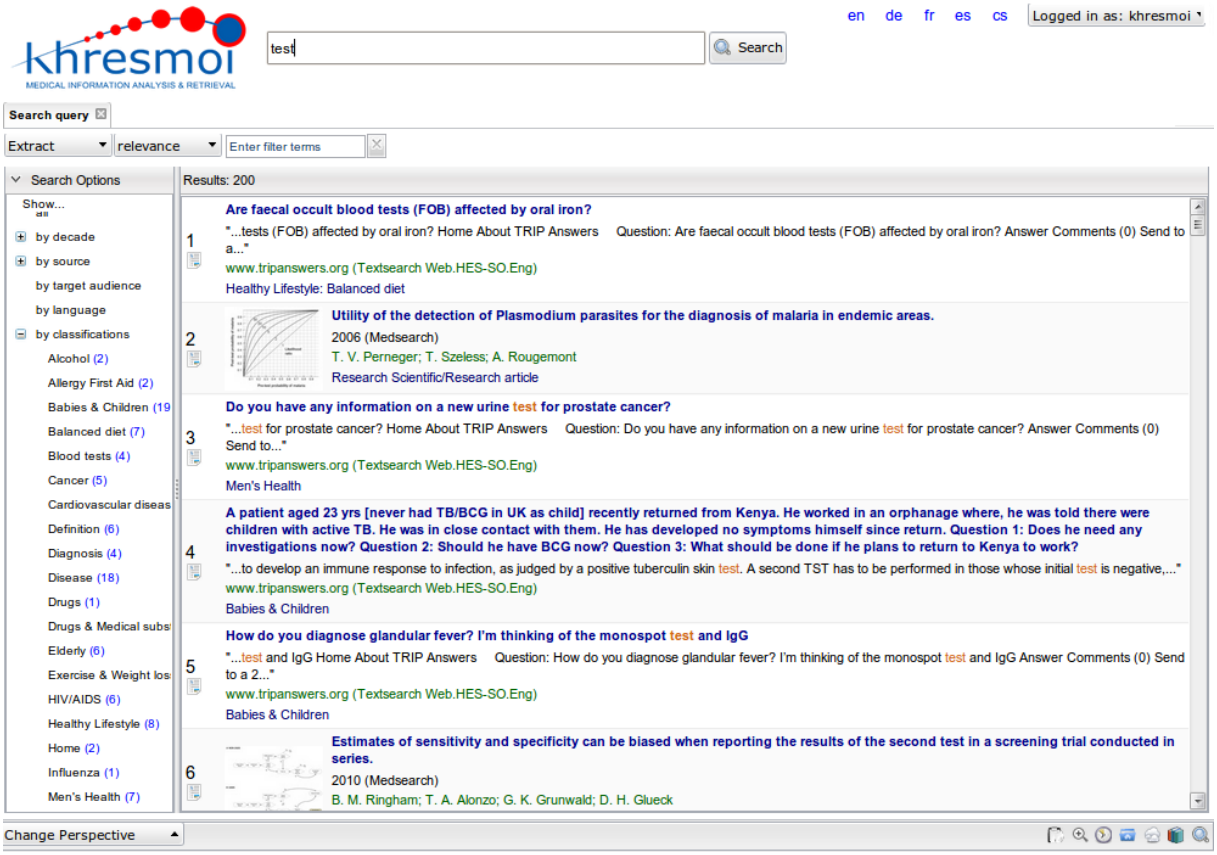

Figure 1: GWT user interface with a simple perspective.

- In the *Personal Library* or *pLib* registered users can create their own portfolio of interesting documents or useful queries. The personal collection can be annotated with tags, and the documents can be filtered, grouped or sorted using the same components that are available for the result view in the Search Tool. Personal documents can be imported from metadata that is available in BibTeX format. Documents in the personal library can also be shared with other users.

- The *Query History* provides a list of past queries (with date, sources searched and number of documents found), which can be re-run from the history, stored in the Personal Library or combined with other queries (thus allowing easy creation of complex Boolean queries). The Query History allows grouping of past queries by date and free-text filtering. For registered users the history of past searches is persisted between sessions.

- The *Detail Tool* shows a preview of important metadata, thumbnails or short summaries for result documents and allows interaction with those documents (such a translating the whole or parts, exporting, printing or saving/bookmarking them). It is described in more detail in Section 4.3.

- The *Tray* can be used for temporary storage of interesting documents or other items during a search session. Unlike the Personal Library, which stores documents of registered users remotely in the Cloud, the Tray is also available for non-registered users.

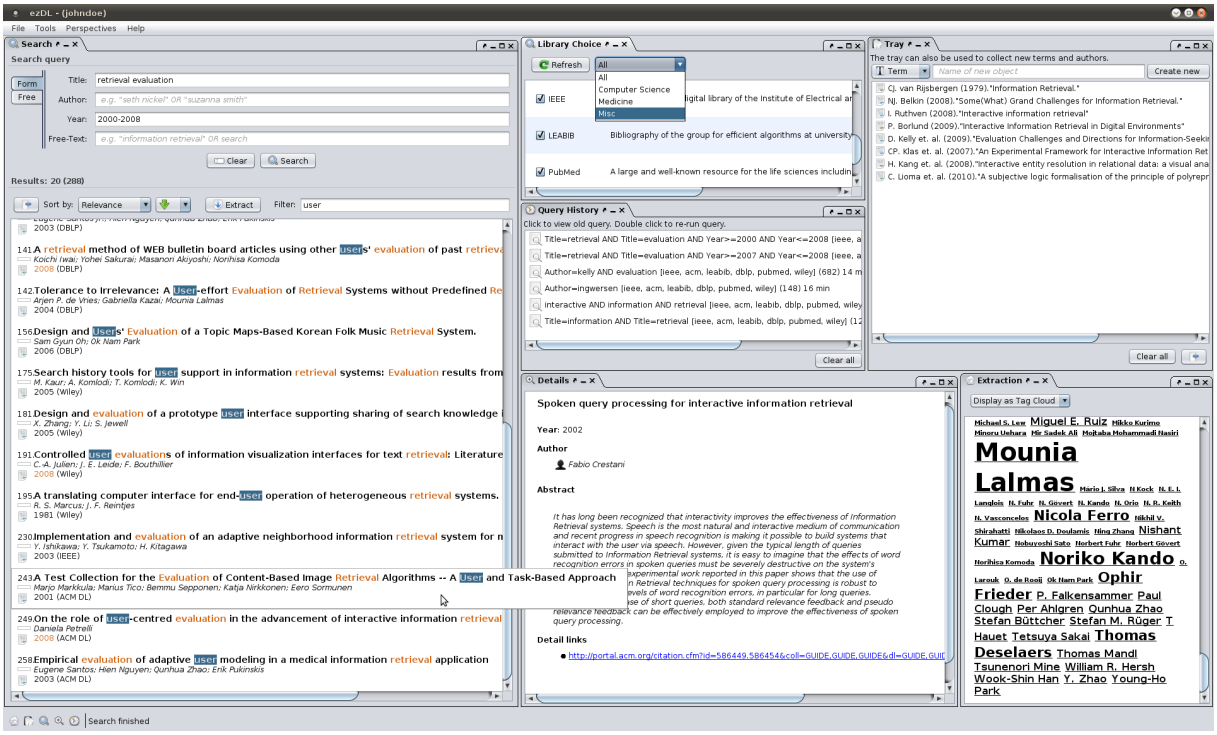

Figure 2: Swing user interface with an advanced perspective.

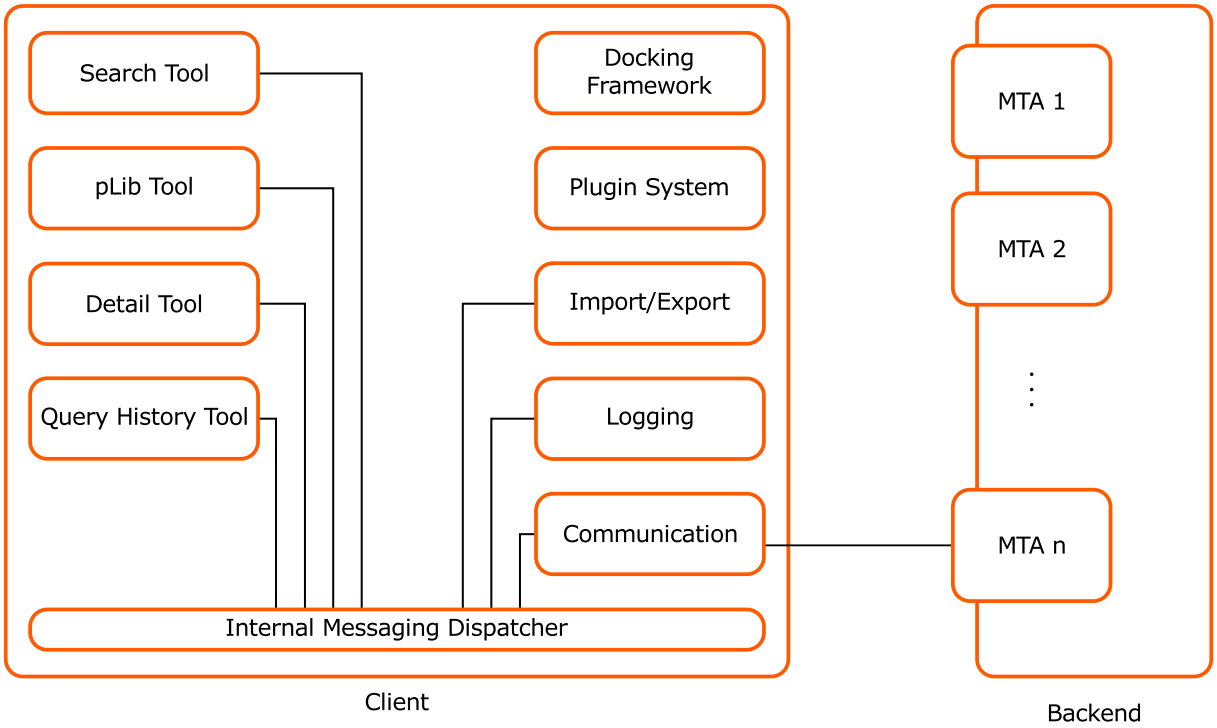

Figure 3: Basic architecture of the *ezDL* frontend.

- The *Group Management* is only available for the Swing user interface and allows for

creation and management of contact or sharing groups. These groups of users make it possible to share a document with a group instead of sharing it with many individual users and can potentially facilitate collaboration or discussions between users. This is being investigated in task T3.3 [12].

- The *Suggestion Viewer* is also only available in the Swing version of the user interface and displays and allows for interaction with tactical suggestions as detailed in Section 3.2.2

Internally, the client uses a messaging infrastructure to facilitate communication between independent components. No components or tools directly call methods of other tools, so in theory no tool depends on the presence of a specific other tool. Instead a tool creates a message which is queued with the internal *dispatcher* and other tools can register and define special handler methods for specific types of messages. Thus it is possible for multiple tools to be registered for a single event. Communication with the backend is likewise handled through messages which are routed by one or more *Message Transfer Agents* [5].

A number of perspectives are predefined for the two current user interfaces. The Swing user interface comes with a Search, an Advanced Search, an Image Search, and an Organizing and Sharing Perspective. The GWT user interface has only the home screen (a single search box), a Search and an Organizing and Advanced Search perspective (which combines the tools of the Advanced Search and the Organizing and Sharing Perspective with the exception of Group Management).

- Search is the basic perspective of the program and for simple ad-hoc searches no interaction with other perspectives is necessary. It contains a simple query form consisting of a single search box, a result list and a detail viewer, which at the start of the session shows helpful search and usage hints.
- Advanced Search additionally provides a view for source selection, and either the tray or the personal library for saving found documents.
- Image Search like the basic Search perspective only contains a search form, a result list and a detail viewer. However the search form is a specialized form for image searches which provides a drop area for images, through which image queries by example, but also relevance feedback on image results is possible.
- Organizing and Sharing allows users to work with the personal portfolio. It has access to the personal library, a detail viewer and the group management.

2.2 UI design principles

In general the *ezDL* frontends (Swing or GWT) follow the design principle of *tiled windows* [29]. The main window is divided into individual tiles that contain subordinate windowlets whose size and position can be changed. The windowlets contain tool views that allow interaction with one of the *ezDL* tools or present information. If a tile contains multiple tool views users can change between the tools using tabs. The tab of the currently active tool view is highlighted.

In addition to the tiled views, the main window also holds a *tool and status bar* and a *menu area*. The menu area contains options to login as a different user (or to register a new account), to change configuration options (e.g. interface language) and to receive help about the interface. In the Swing desktop application the menu area takes the form of a common menu bar and also holds menus for file operations (importing from and exporting to a file, quitting the application), opening new tools and manipulating the perspectives. In the GWT frontend the menu area is located in the upper right part of the browser window (see Fig. 4).

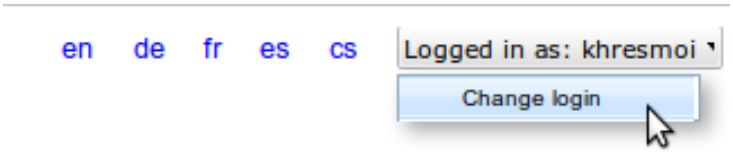

Figure 4: The menu area for the GWT frontend (language switching and login).

The tool and status bar is located at the lower border of the frontend and shows icons for currently active tools as well as status messages for the system.

Figure 5: Tool icons in the tool and status bar (GWT frontend).

2.2.1 Drag and drop

Results and other list items in the *ezDL* frontend are represented by draggable interface objects. These information objects all bear an icon that denotes the type of object it represents (e.g., website, image, term). By dragging and dropping the information objects onto other tool views it is possible to interact with the system using direct manipulation.

Examples for the use of drag and drop are:

- one or more results are dragged to the personal library or to the tray to store them
- an old search from the query history is dragged onto the search form to rerun the query (or to combine it with the current query)
- an image from the detail view is dragged onto the image search form to add it as a new search condition
- an extracted search term or author name is dragged to the tray to collect them for future searches
- the tray is used to collect possible search terms which are then dragged to a query field to add them
- a user is dragged to a contact group to add them

2.2.2 Context menus

In addition to drag and drop the system should provide context menus for all objects which allow additional actions. The actions in the context menu will always work on the currently selected object (see Fig. 6).

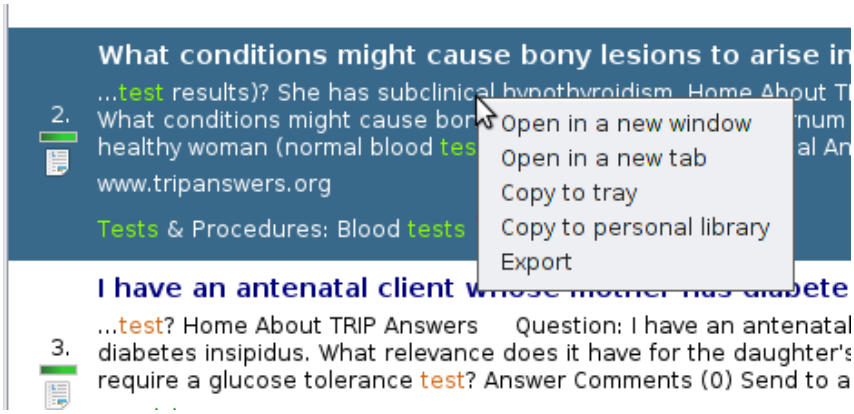

Figure 6: Context menu for a result item.

2.2.3 Input prompts

All text fields that are presented to the user are prefilled with input prompts that either show an example of a valid input or tell users what they should do or type at this prompt (see Fig. 7). This helps to make the user interface easier to learn and understand. Unlike input hints which are to be placed outside of the text field, input prompts can not easily be missed or ignored by the user [29]. Where necessary input prompts are combined with input hints.

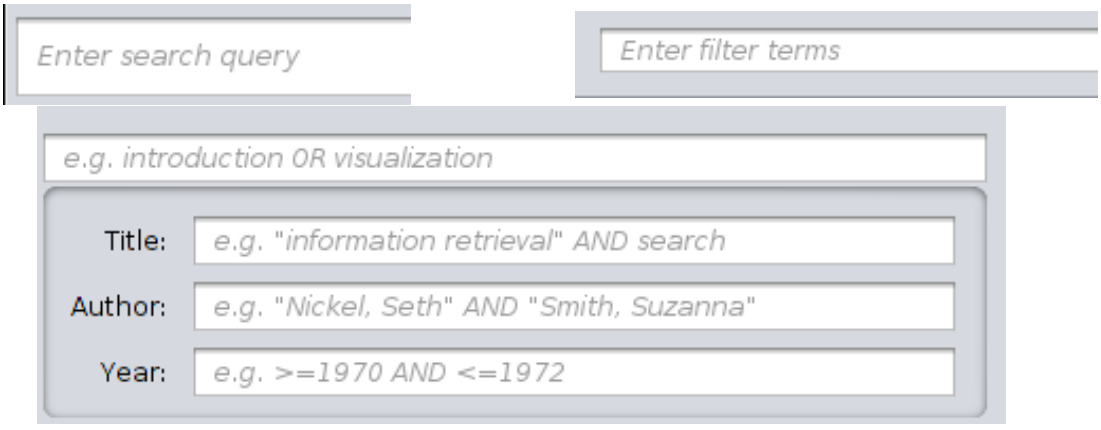

Figure 7: Examples of input prompts.

3 Support for query specification

At the core of the Khresmoi user interface lies *ezDL*, a versatile search system that can be used in many different scenarios. A number of different search or usage scenarios are currently supported, centered around the type of queries issued.

Some users prefer simple text-based queries using the well-known paradigm of a single search box (see Fig. 8 and 9), while other users (e.g. radiologists) want to search based on example images which cannot be described textually (see Fig. 10). For this reason, multiple different query forms are implemented and available in *ezDL* at the same time. Each has its own advantages and disadvantages, so it is possible to use multiple query forms at the same time. A query entered in one form is displayed in all query forms that support displaying the query: entirely text-based queries can be edited in all currently implemented query forms (e.g., in an advanced query form using fields for different facets of the result documents, as seen in Fig. 11); queries containing images (as positive or negative examples) can be manipulated only in the image query forms.

Figure 8: Simple text based search form (Swing user interface).

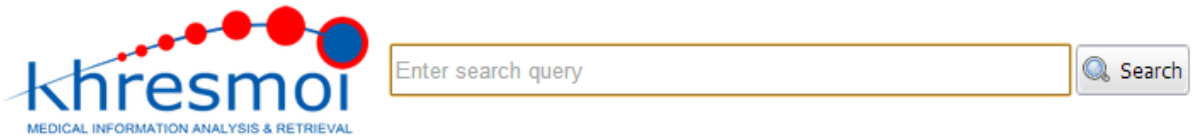

Figure 9: Simple text based search form (GWT user interface).

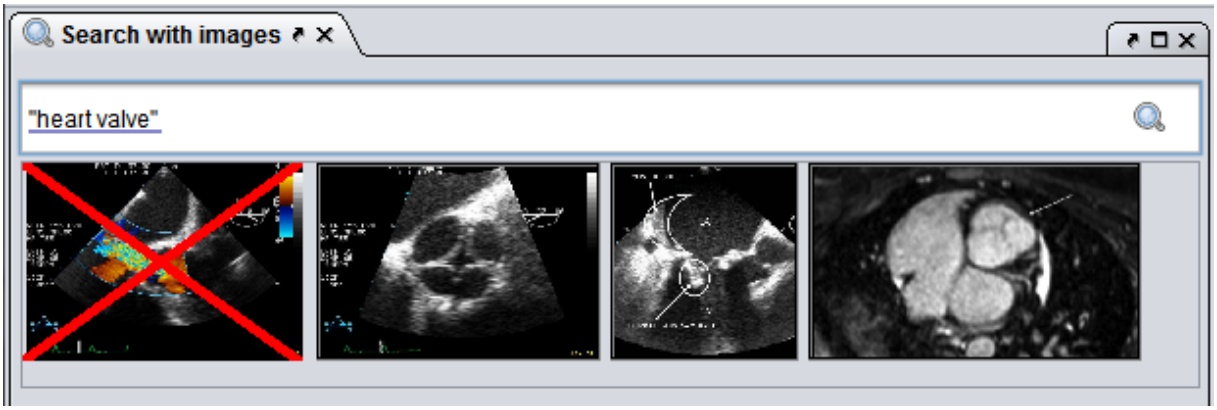

Figure 10: Search form using text and images (Swing user interface).

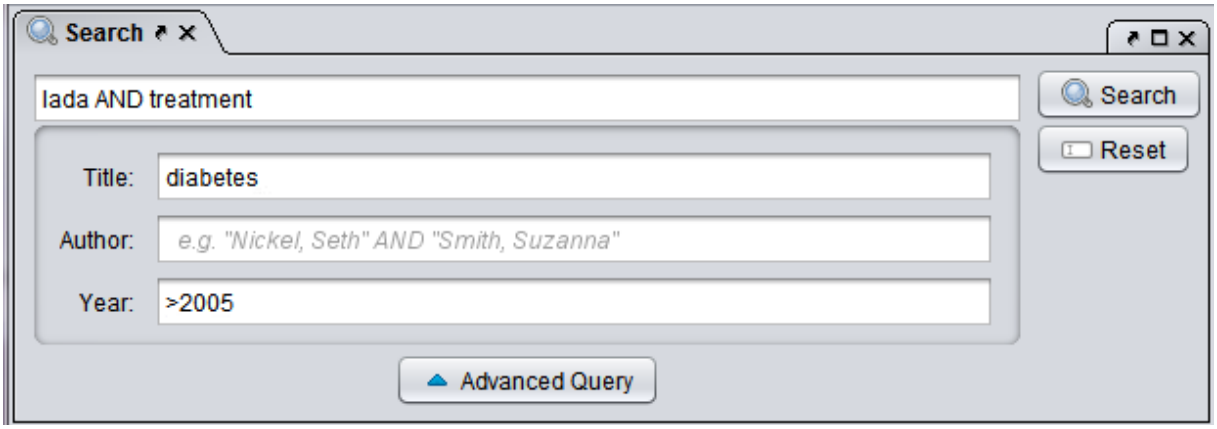

Figure 11: Advanced search form with fields (Swing user interface).

3.1 Search queries

In *ezDL*, multiple different ways of representing a query are used: the searcher uses one syntax for specifying what she wants to find, *ezDL* uses an internal representation of that input and each search system the system is connected to uses its own query syntax (image search, textual search, semantic search, etc.). For this, each query representation has to be converted into other representations. Figure 12 gives an overview of the transformations that occur between the user's input and the query that is sent to the IR system. First, the user enters a query in the user interface. The query is parsed into the *ezDL*-internal representation using the grammar detailed in Appendix A. This internal query is transferred to the back end of the system where it is routed to one or more software components (called "wrappers") that convert it to the query syntax a specific remote resource (or Digital Library, DL) understands, forming a *DL query*.

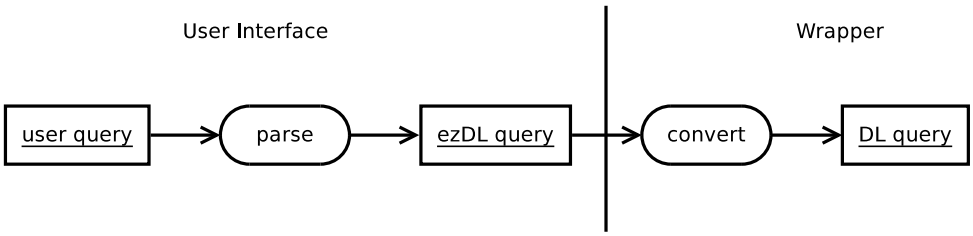

Figure 12: Query data flow.

The *ezDL* architecture does not force any single syntax for query specifiction. All query-processing components use a generic translation between the user input and the data-structure internally used for representing the query. The syntax described in this document is one possible syntax supported by the user interface that covers many common operators.

3.1.1 Internal query representation

The internal query representation is a query tree, implemented using the composite pattern. A class diagram of the structure is shown in Figure 13.

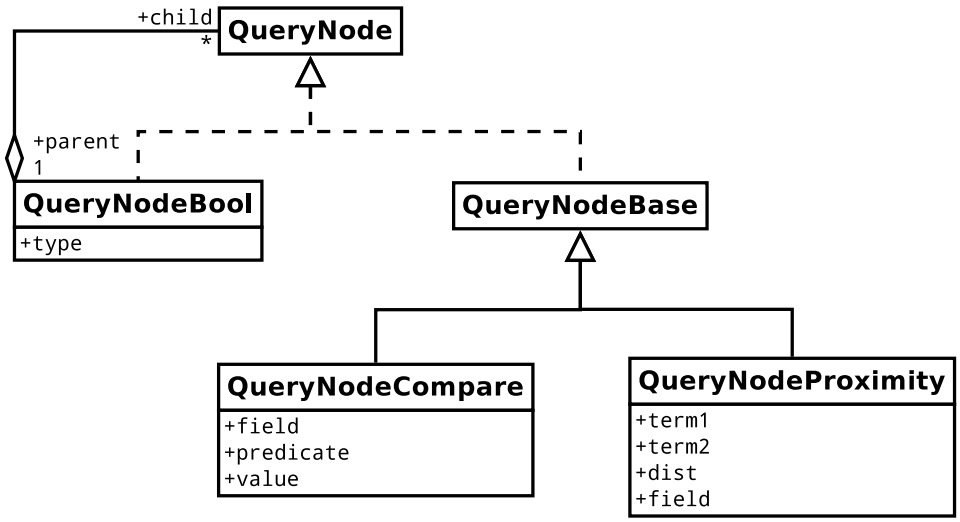

Figure 13: Query representation.

The `QueryNodeBool` class represents a boolean node – either `AND` or `OR`. The boolean nodes can have children that are either further boolean nodes or actual condition nodes. A query node can also be negated.

There are currently two nodes that represent conditions: the classes `QueryNodeCompare` and `QueryNodeProximity`. The former represents a simple comparison of a document's field with a value. It may stand for the conditions like "the title field contains the phrase 'eye tracking' " and "the publication year is 1999 or later".

The proximity nodes contain the condition that a given field – e.g. the title – contains two terms within a given maximum distance. An example for such a condition is "the title field contains 'information' and 'retrieval' at most 2 words apart." This is useful if the searcher wants to be robust with respect to wording.

Since comparisons are not always text-based, the conditions are checked against `Matchables`. A `Matchable` is a way to store the object of a comparison. Examples for those are `StringMatch`, `PersonMatch` and `ImageMatch`. A `StringMatch` contains a string of characters to search for – this is used in basic queries like `"information retrieval" AND evaluation`. `PersonMatch` contains pre-parsed information about a person name. A `PersonMatch` stores the first name and last name parts seperately. Thus, classes that need only the surname part can get them from the object without needing additional knowledge about what the user typed and what she meant by that. An `ImageMatch` object contains an image to search for, so classes that process the query and are able to process image queries can work with them, while classes that don't can ignore this part of the query.

3.1.2 Query syntax

In *ezDL*, the user is offered a rich syntax for formulating queries in text form. The ANTLR grammar used in the interface is given in the Appendix.

Queries may be a simple concatenation of terms similar to the queries usually issued in web searches:

```
diabetes treatment children
```

In this form, each term is connected by implicit `AND` operators and equivalent to the following query:

```
diabetes AND treatment AND children
```

Matches:
“Treatment of diabetes mellitus in children”

Does not match:
“Diabetes mellitus in children”
“Treatment of diabetes insipidus”

Phrases are enclosed in double quotation marks. The following query searches for documents that contain the terms “diabetes” and “mellitus” without any terms inbetween and in the given order:

```
"diabetes mellitus"
```

Matches:
“Treatment of diabetes mellitus in children”

Does not match:
“Treatment of diabetes insipidus ”
“Diabetes insipidus and mellitus”

Alternatives can be connected by the OR operator:

```
diabetes AND mellitus OR insipidus
```

Matches:
“Treatment of diabetes mellitus in children”
“Treatment of diabetes insipidus in children”
“Diabetes in children is most often mellitus”

Does not match:
“D. insipidus ”
“D. mellitus ”

The `OR` operator has a higher precedence than the `AND` operator to reduce the need for parentheses in queries that contain synonyms connected by `OR`.

Proximity search is done using the proximity operator:

```
diabetes /2 mellitus
```

Matches:
“diabetes mellitus”
“diabetes insipidus and mellitus”

Does not match:
“diabetes insipidus and also mellitus ”

Precedence of query parts can be indicated by using parentheses as usual:

```
(diabetes AND mellitus) OR insipidus
```

Matches:
“Treatment of diabetes mellitus in children”
“Diabetes in children is most often mellitus”
“Treatment of diabetes insipidus in children”
“Treatment of insipidus in children”
“D. insipidus ”

Does not match:
“Diabetes”

In the above forms, queries operate on a default field. Documents are organized in fields: the title is a field, the text is one and also the abstract. To search in a specific field, the field is given in the notation “field=term”. E.g.:

```
Title=diabetes
```

Matches:
“Treatment of diabetes mellitus in children”
“Diabetes in children is most often mellitus”
“Treatment of diabetes insipidus in children”

Does not match:
“Diabetes” in the abstract

If no field is explicitly stated in the query, the default field is assumed. The field used as the default field depends on the query form and its context in the user interface. In a form that has query input fields for title and text, the default document field in the title query input field would be title. In this field a query `diabetes` would refer only to the title field of documents. If a different field is meant, this can still be given explicitly. So even in a query input field that is about the title, users can search for `Author=Smith`.

In cases where multiple query parts refer to the same field – or when using a proximity operator, a whole subquery can be stated to be about a field using curly braces:

```
Title=retrieval Author={Fuhr OR Kriewel}
```

Matches:
Title: “retrieval”, Author: Fuhr
Title: “Information retrieval”, Author: Kriewel
Title: “Information retrieval”, Author: Kriewel and Fuhr

Does not match:
Title: “Information search”, Author: Fuhr
Title: “Information retrieval”, Author: Smith

Simple terms (not phrases!) can be written using wildcards in cases where the exact spelling is unknown or unimportant and it is uncertain if the remote service properly stems the search terms. The `$` character is a placeholder for exactly one single character; `#` is a placeholder for none or many unknown characters:

```
retr$$val eva#
```

Matches:
evaluating retrieval
retreival evaluators
But also:
retrooval evangelist
Does not match:
retrival evaluation
retreival test
retrieving evaluations

In searches for person names (e.g. in the document fields “author” and “editor”), the user can specify which parts of a name belong to the surname using the comma notation in the form “Lastname, first name” in combination with the phrase notation: `Author="Spärck Jones, Karen"` (all parts given), `Author="Spärck Jones,"` (only last name parts are known or interesting) or `Author=", Karen"` (only the first name part is known).

It is then up to the wrapper that queries a remote resource to decide how the submitted name parts are used to form the query.

All these features can be combined into complicated queries like this one:

```
Author="Fuhr, Norbert" AND Title={retr$$val eval# OR test# OR exper#}
  AND Text=interactive
```

3.1.3 Translation between internal and remote query languages

Each query is represented in the form shown in section 3.1.1. Remote resources such a Mimír index may have a vastly different syntax.

To this end, the internal query is converted to a query that the remote resource can work with. This is usually done by traversing the *ezDL* query tree and translating the query bottom-up into the destination language. In cases where the *ezDL* semantics of a part of the query cannot be translated exactly into the remote system's language, a choice has to be made between translating part more loosely (resulting in more hits, but also more irrelevant ones) or more narrowly (resulting in fewer hits, but also fewer irrelevant ones).

Some remote resources might provide query languages that do not contain Boolean operators. Here the difficulty lies in translating the structure of the whole query. For this reasons, wrappers can operate in a mode where they first transform an internal *ezDL* query into disjunctive normal form (DNF). Each conjunction is then submitted individually to the remote service and the result lists merged.

3.2 Suggestions

Searchers can need assistance at various times during their interaction with an information system. Since they have a lack of knowledge about a topic which drives them to search, they often don't know how to specify their information need. Belkin calls this the *Anomalous State of Knowledge* [6]. Lay people in particular are often unfamiliar with the specific medical or technical terms which may be used in the document they are looking for. They have only a vague understanding of their information need and cannot express it with words. Searchers in general often cannot anticipate synonyms or word choices which document authors might have used instead of those the searchers themselves may be thinking of.

In addition to problems during query formulation, which come from a lack of domain or terminological knowledge, non-professional searchers very often also lack in procedural or search knowledge. Even though search systems might offer a large amount of helpful functions, many end users don't know how to employ them successfully to find relevant documents. As a counter for these problems three different but complementary support and suggestion functionalities were developed for the Khresmoi user interface. They help with query formulation, provide suggestions for search continuation after the user has already issued a query and seen some results, and can help a searcher with structuring or planning a search.

3.2.1 Query suggestions

To help users with some of the issues during query formulation a subsystem for observing the process was integrated which can make suggestions to change the search query.

The system is based around a ProactiveObserver and extends previous work on proactive support [15, 25]. The observer waits either for an explicit call or observes pauses during the typing, to run a set of modules. Each module checks the tokens of the query (one or several) for possibly helpful suggestions. If suggestions are found by any or all of the modules those are gathered and presented to the searcher in form of a dropdown list. The items in the dropdown list can have additional explanations which are shown when the users hovers the mouse pointer over an entry or selects an entry with the cursor keys (see Figs. 14 and 15 for different realizations of this features).

Each proactive suggestion module provides query suggestions of one specific type and the dropdown list of query suggestions can contain a mix of types. Various presentation and trig-

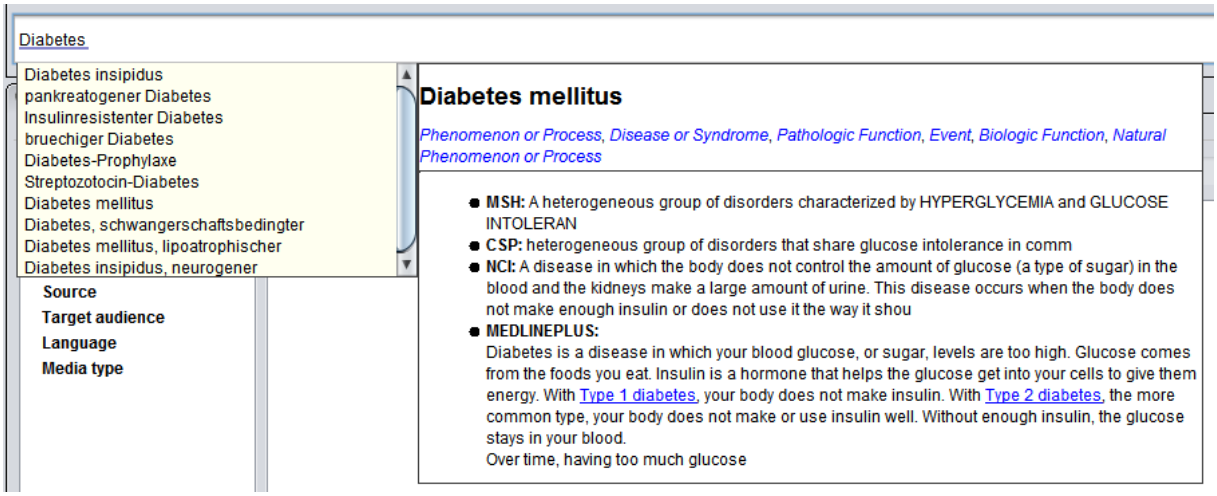

Figure 14: Disambiguation with explanation (Swing user interface).

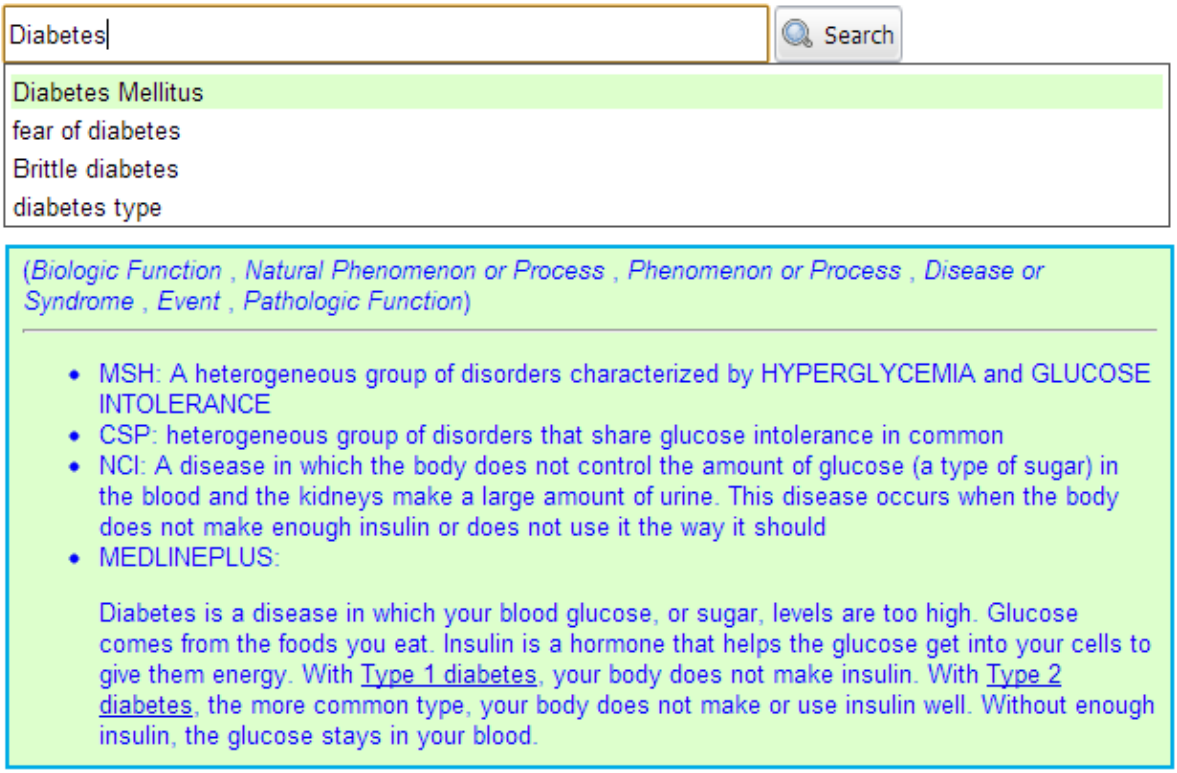

Figure 15: Disambiguation with explanation (GWT user interface).

ger options for the suggestion dropdowns were explored and the results have been reported in deliverable D3.2 [4, 13].

Implemented modules which are available in the Khresmoi search system or can be integrated are

- a spell checker which suggests spelling corrections based on the user's preferred query language (using a service developed in task T4.1a and shown in Fig. 16),

- a translation module which suggests possible translations for query terms into the target language of the selected document selection (using a service developed in task T4.5),
- a module which suggests related terms and synonyms to expand the query with (related terms in this context are provided by a service that uses co-occurance to determine relatedness),
- a module that suggests query terms based on queries previous users typed,
- a module that provides disambiguation suggestions based on an ontology (using a service developed in task T5.1 and described in deliverable D5.2 [24]).

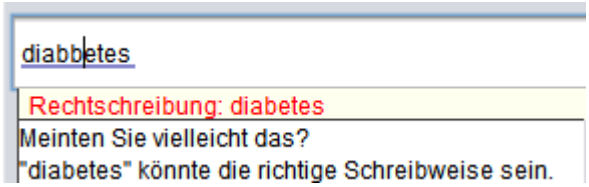

Figure 16: Spelling suggestion with short explanation.

The subsystem is easily extendable. The processing of terms is located in modules which can be registered with the proactive subsystem at runtime. Additionally it is possible to use the plugin system of *ezDL*, which will be described in more detail in deliverable D3.6 on the user interface framework. In this case, a single module is compiled as an OSGi bundle and provided to users as a download (in a similar way to browser extensions). The plugin system announces the module at the start of the user interface. This allows easy customization of the types of suggestions and their presentation by search system administrators or end users.

3.2.2 Strategic support

The strategic support for the Khresmoi user interface builds on work done by Kriewel [17, 18, 19, 20]. Many studies have shown that inexperienced searchers are often employing inefficient, and sometimes even counterproductive search tactics when reformulating their queries [9, 22]. A search system which can point out the most productive moves in a specific situation can help users to find more relevant information faster. By explaining how these suggestions can help, the system can also further the searchers' understanding of successful search strategies.

A suggestion subsystem using cased-based reasoning (CBR) was integrated into the Khresmoi prototype (but will not be part of the first round of the user-centred evaluations). It is described in detail in [28]. The system is triggered automatically after each search and uses query terms, fields, and operators, as well as information collected from the search results to construct a case that describes the current search situation. This case is compared with a database of previous cases, and solutions in form of search suggestions are derived from the solutions of those previous cases. The suggestions are presented to the user on demand (see Fig. 17) but the best suggestion for the current situation could also be used automatically to improve a unsatisfying search result. Searchers can rate suggestions as useful or not useful, or execute them with a button click.

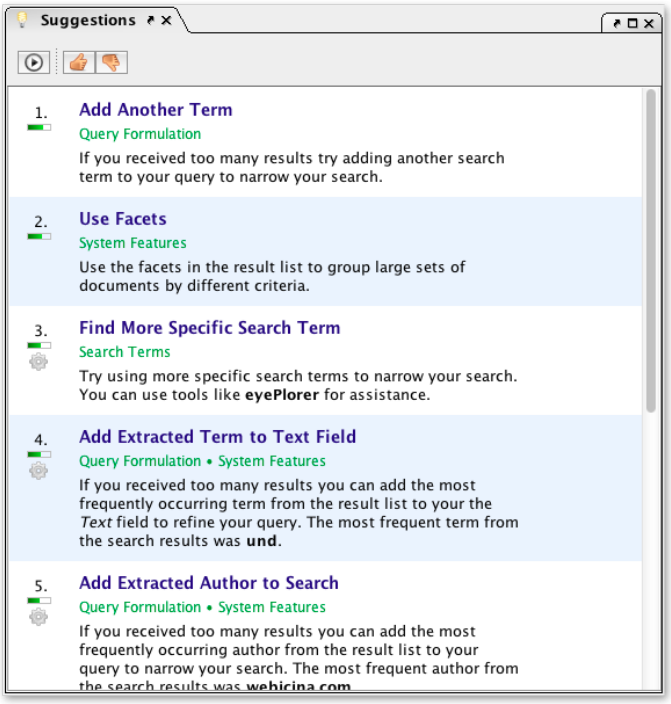

Figure 17: Search continuation suggestions.

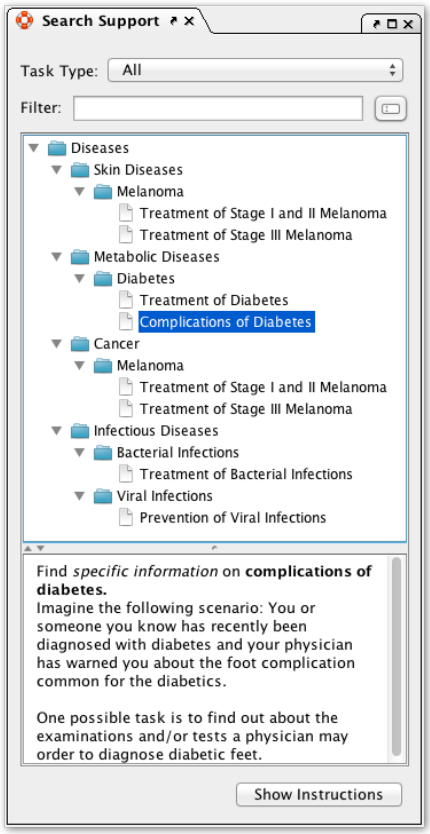

Figure 18: Scaffolding for common search task.

The suggestions (or case solutions) are common search tactics or strategies from the information science literature adapted to the system and domain. Table 1 shows the currently implemented search suggestions which have been categorized according to the classification system for moves, tactics, stratagems and strategies originally proposed by Marcia Bates [1, 2, 3].

The similarity between two situations (or cases) is a weighted sum of the similarities between the two cases' attributes. Similarities between attributes are computed using the following formulas (depending on the type of the attribute):

- Similarity of numerical attributes

$$sim_{num}(k1,k2) := \frac{min(k1,k2)}{max(k1,k2)} \tag{1}$$

- Similarity of sets

$$sim_{set}(t1,t2) := \frac{|t_1 \cap t_2|}{|t_1 \cup t_2|} \tag{2}$$

- Similarity of weighted sets (e.g., terms with frequencies)

$$sim_{wset}(t1,t2) := \frac{t_1 \cdot t_2}{|t_1|\,|t_2|} \tag{3}$$

- Similarity $sim_\Gamma(a,b)$ of two situations a and b

$$sim_\Gamma(a,b) := \frac{\sum_{k=0}^{N} w_k \cdot sim_k(a_k,b_k)}{\sum_{k=0}^{N} w_k} \tag{4}$$

Situational search suggestions of appropriate moves can be combined with scaffolding techniques that help searchers tackle complex information tasks, such as searching for complete information concerning a specific diagnosis. Information problems encompassing several aspects are often difficult or impossible to answer with a single document or even a single query result, but require that the task be split into several subtasks. With the help of software-realized scaffolding the system can provide a guided search support for the domain of medicine and health-related search tasks [7, 8]. Query suggestions and the situational suggestions are included into this larger context. Guided searching can help inexperienced searchers with complex problems to divide their information task into smaller subtasks and use the tools of the search system to plan and organize their search (see Fig. 18). Through the use of scaffolding they can learn to employ proven strategies such as pearl growing [26].

3.3 Use of semantic information

Semantic information can be used in several ways during query specification. One realized option is the provision of a disambiguation service in the form of suggestions during query formulation. With the help of definitions and explanations, searchers can pinpoint a specific meaning of a homonym, heteronym or polyseme. This disambiguation information is used when querying the Mimír index, where documents have been annotated with semantic information.

	Stratagems	
Search With Extracted Author	Execute a search for the author with the most publications in the current result list (who was not part of the current query) to find other relevant documents	Author Search
	File Structure Tactics	
Browse Facets	Use the search facets to subdivide large result list	CUT
	Term Tactics	
Add Phrase(s)	Narrow search by adding phrases to the current query	RESPACE
Remove Phrase(s)	Broaden search by removing phrases from the current query	RESPACE
Add Another Term	Narrow search by adding another term to the current query	
More General Term	Broaden search by finding a more general term with a thesaurus	SUPER
More Specific Term	Narrow search by finding a more specific term with a thesaurus	SUB
Vary Spelling	For example, vary spelling to compensate e.g. for differences in British and American English	RESPELL
	Search Formulation Tactics	
Add Extracted Author	Narrow search by adding the most extracted author from the result list who was not part of the current query	
Add Extracted Term	Narrow search by adding the most extracted term from the result list to the current query	
Remove Author(s)	Broaden search by removing one or more authors from the current query	REDUCE
Remove Term(s)	Broaden search by removing one or more terms from the current query	REDUCE
Search in Recent Years	Narrow search by restricting the query to the last few years	
Use Text Field	Broaden search by using the Text field instead of the Title field	
Use Title Field	Narrow search by using the Title field instead of the Text field	
	Idea Tactics	
Extract From Clipboard	Generate new search terms by extracting from documents in the clipboard tool	
Extract From Results	Generate new search terms by extracting from documents in the current result list	

Table 1: Implemented search suggestions, classified according to [3]

In addition to the original query term, the interface sends the disambiguation label and the URI of the concept from the ontology. This can be used by the *Khresmoi Mimír Interface* (KMI) to rewrite the query so as to cut away irrelevant documents (that contain a homonym or polyseme of the concept), while at the same time adding relevant documents which are annotated with the concept but use a different term than the one provided by the searcher.

Figure 19: Disambiguated terms are highlighted.

After a user has chosen a specific meaning, the term is highlighted within the query to mark this fact (see Fig. 19). The term can still be edited, but in that case the highlighting is removed and the searcher would have to choose a new (or the same) disambiguation.

Another way to use semantic information during the process of query specification is to provide a way to directly specify concrete values for selected classes, which have been used in the process of annotating the indexed documents. E.g., an interface for medical professionals could have dropdown boxes to allow them to specify that a result document should cover diagnostic procedures for a disease, and deal with adult women. This information can be combined with the query terms provided by the searcher to narrow down the search (see Fig. 20).

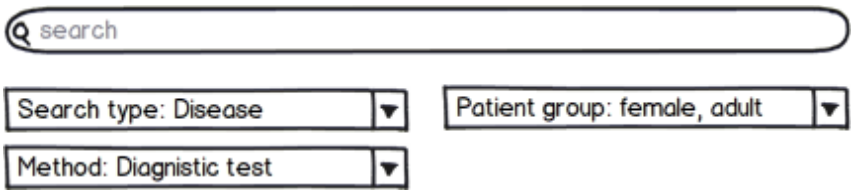

Figure 20: User interface mockup with dropdowns to specify annotations.

Further narrowing down is possible through facetted browsing within the result set using some classes of annotations as metadata that provides the facets and values. Facetted browsing can be implemented in the form of a filter on the result set, where only those elements that match the current facet value are shown (see Fig. 21), or in form of grouped lists, where all results are shown in a single list that has been divided into several sublists and the selection of a facet value will jump to the sublist that contains all results that match this value (see Fig. 22).

Another possibility depends on the presence of relations between classes in a domain ontology. When a concept from the ontology is identified during query specification, e.g., the name of a disease, the user interface can use information from the ontology about possible relations with other concepts to provide users with suggestions on how to specify their query using semantics (see Fig. 23).

A more general research on the usefulness of ontologies in medical search was started as part of an ongoing PhD thesis [23]. The research tries to combine the semantic annotations of multiple documents with the information of the ontology to improve search. First findings showed that ontologies in medicine achieve a good term coverage at both document and query level and were able to improve ontology-based search compared to conventional full text search (when considering number of documents found and number of queries without results).

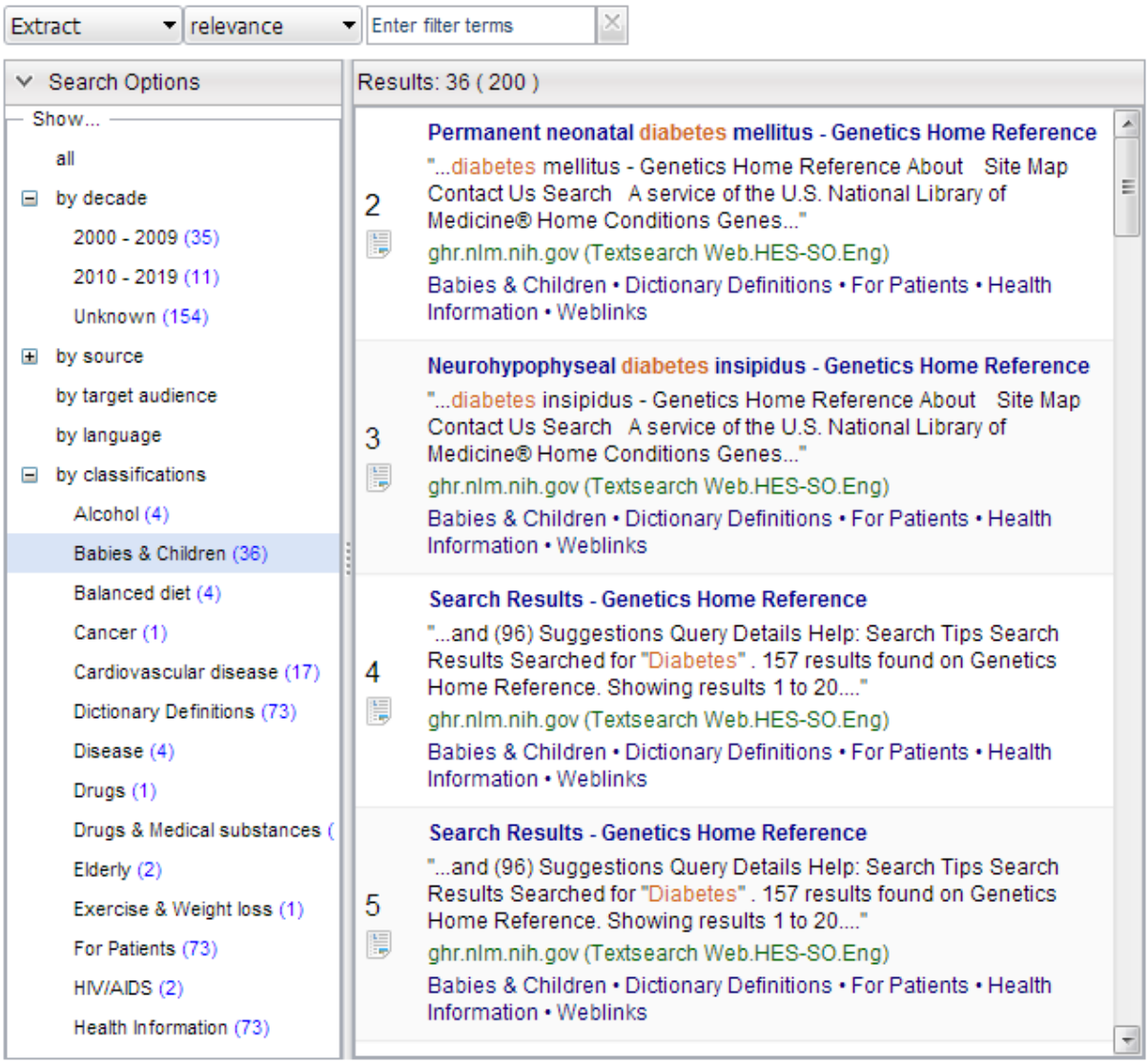

Figure 21: Filtered list for facetted browsing (GWT user interface).

4 Support for result presentation

Presentation of a result set has strong effect on user satisfaction and search success. While presentation style has less effect on the outcome of a search for simple or fact based queries it is more important when dealing with complex information needs. For resolving the latter a user may be required to view several documents, to work with a very large result set, to make connections between result items or to use insights gained from the search to issue related queries. Users can also benefit from non-textual document representation in different search contexts [14]. Result sets in the Khresmoi search system are presented in such a way that required operations can be performed easily, allowing the user to focus on the task at hand. This section describes two different layouts for result sets, gives more specific information about the document representation and details the main features of the result presentation component of

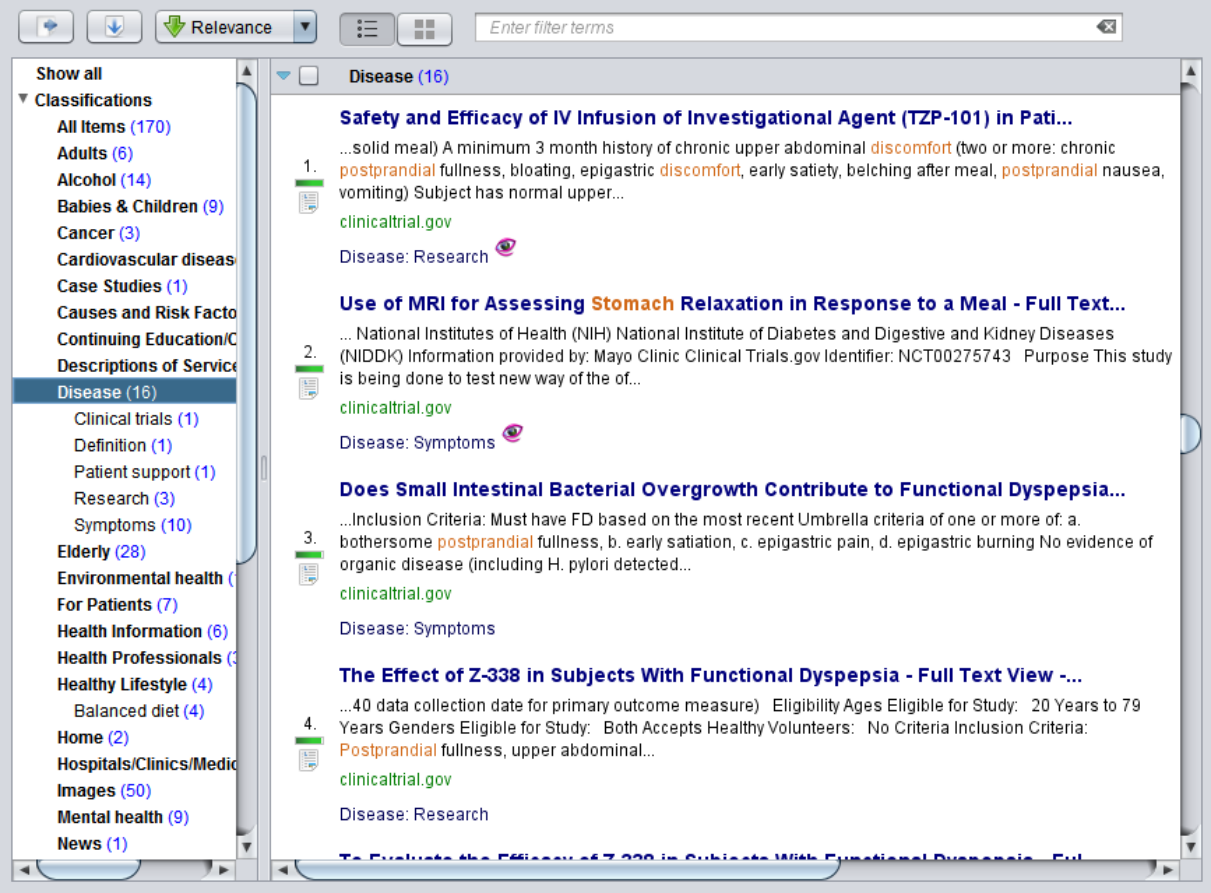

Figure 22: Grouped list for facetted browsing (Swing user interface).

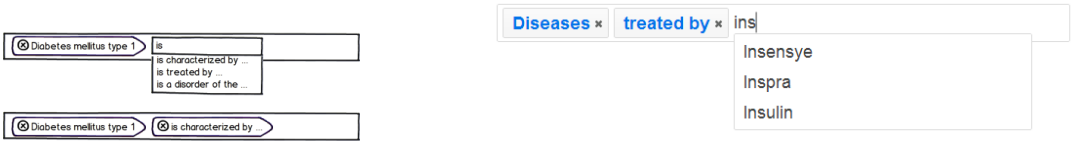

Figure 23: User interface mockup using relations between concepts.

the Khresmoi user interface.

4.1 Result set layouts

Two result set layouts have been created, both suitable for different search situations. In the list layout (Figure 24) every single result item is displayed as an entry in the list. Every entry consists of different—mainly textual—features. For a detailed description of the item layout see Section 4.2. If the text in a list entry is too long to be displayed in the current screen configuration, an overlay is shown if the user hovers the mouse over the entry. An example of this overlay is visible in Figure 24. Document details can be retrieved by a single mouse click on any result item.

The grid based layout (see Figure 25) is mainly for image documents. Result items are

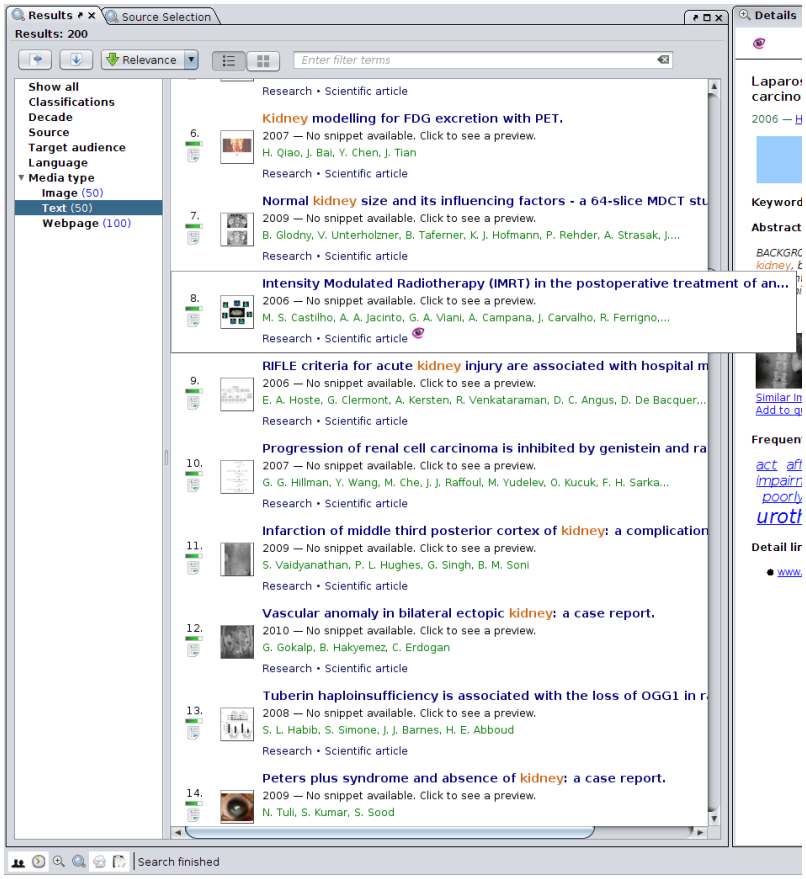

Figure 24: A result set in list view.

displayed as tiles in a grid, i.e. every row in the view contains more than one element, depending on the screen configuration. Every image is represented by a small 80x80 pixel thumbnail. More information can be displayed in the grid tiles, like an image description or document meta data, if available. The detail view works equivalently to the list based design and is trigged by a mouse click on a tile. The grid layout allows the user to gain an overview of the result set quickly as it reduces the necessity for scrolling.

Both layouts feature various interface elements that allow users to interact with the result set. A full text filter lets searchers drill down into a large result set without requiring them to issue a new search. All user entries are matched against the information that was retrieved with the original query response. Document details are not included, as they have to be explicitly requested by the user and are shown in a separate detail view. Furthermore, the results can be sorted according to different criteria, like document creation date and relevance to the query. The system also features a faceted browsing mechanism. User can select a document subset by clicking any of the facet labels on the left side of the result view. All subsets are visually grouped in the result view and can be hidden with a single mouse click.

Moreover, advanced features like the extraction of keywords and authors of documents are obtainable. The extracted terms are displayed in a separate cloud view. The user can also choose to export documents meta data in different formats for later use, like importing the data into a citation management tool.

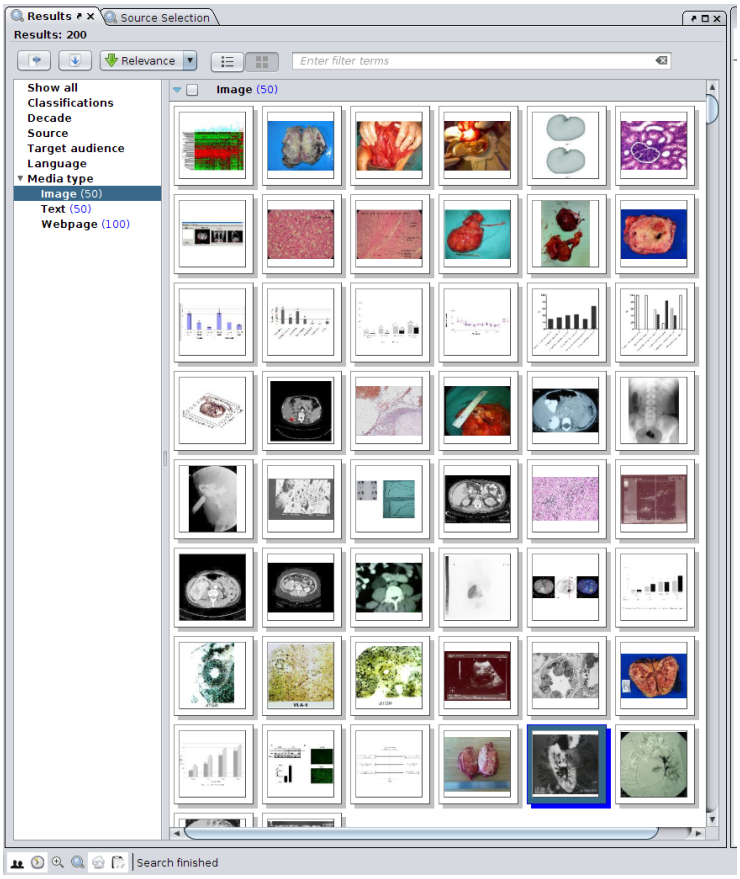

Figure 25: A result set in grid view.

Regardless of the result set visualization, clicking an item will always make the system show this item in the detail view. All available information corresponding to the item is displayed there, including e.g. links to full text documents.

4.2 Result items

As described in Section 4.1 the user interface features two distinct views of result sets. For the list layout various styles for visual representation of result items have been designed. These so called *renderers* differ in content and layout. In deliverable 3.2 [4] a user evaluation was described which aimed at finding suitable meta data elements to include in the renderers. User preference was also measured. The results of this study lead to the current renderer set up.

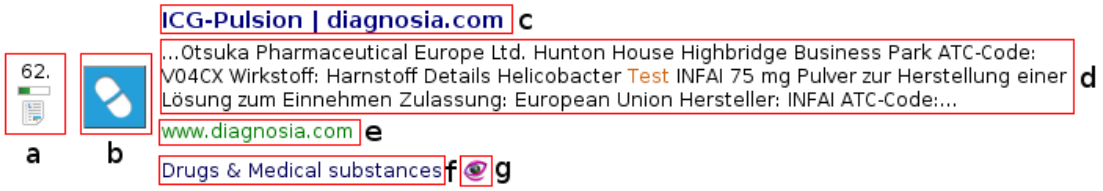

Figure 26: A single result item in the list view (with query term ‘test’ highlighted).

Different document types require different renderers as each document type may feature other meta data elements. Therefore, the result view can display various renderers in the same list according to the respective document type. All renderers allow for highlighting of matching query terms in the title, snippet or keywords of the result item. In Figure 26 the renderer for the type *text document* is shown. The visual elements are:

- a: General information about the document, i.e. rank in the result set, relevance bar and document type shown as an icon
- b: A thumbnail of an image contained in the document (optional)
- c: The document title
- d: A short document snippet or description
- e: The source URL or document author
- f: Keywords that categorize the document (optional)
- g: Icons that represent actions the user has performed on this result item, here viewing the document details

The grid view uses three different result element visualizations. For image documents a single thumbnail is shown which can be seen in Figure 27. If an image description is available it is also visualized as shown in Figure 28. Text documents are rendered similarly to the list view but with a larger thumbnail (depicted in Figure 26.b) replacing the ranking information (depicted in Figure 26.a).

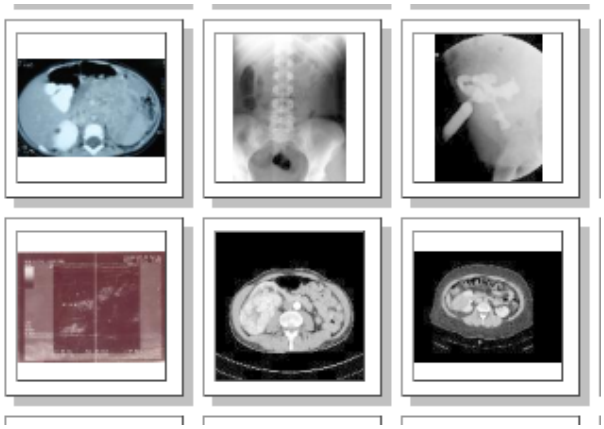

Figure 27: Some result items in grid view.

In the next section the detail view is introduced as a means for a more thorough inspection of result items.

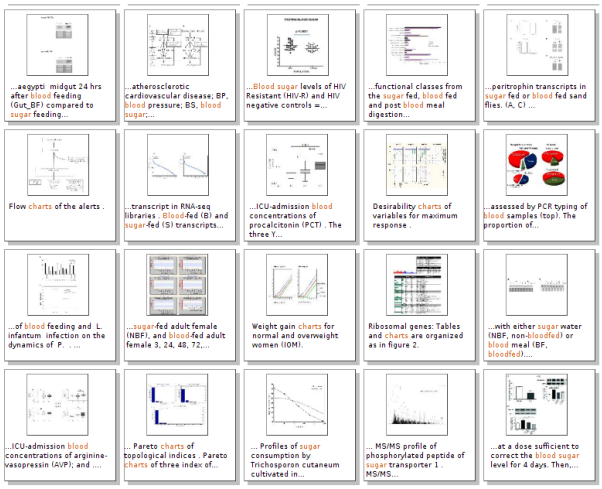

Figure 28: Some result items with image description in grid view.

4.3 Result previews

Different tasks, user groups and content types have different demands on the presentation of results. Some tasks are quick fact-finding problems, which are often answered by a single sentence or two, that is then verified by looking at a few other sources. Other tasks make it necessary to look at a large number of results to find several documents which have to be read in detail. The result presentation in *ezDL* is modular to allow administrators to pre-configure which parts to display, or users of the system to personalize the display themselves. In addition, different result types (like webpages, images, scientific articles, study descriptions, term definitions, etc.) use different renderers, so that they can be configured independently.

Fig. 29 shows an example of result details for a web article. The actual result view can be configured to show only some parts of the presented information. The top area presents icons that show which actions were already performed on this result document, as well as a number of action buttons. In the example shown, the searcher has viewed, saved and printed the document (1). The buttons allow to navigate to previously viewed results (2), opening the document in a new window, saving the document in the personal library of the search system, printing it, and exporting it in various formats (3).

The information shown for each result includes the title of the webpage (4), a clickable URL (5) which leads to the full version of this document, a number of assigned keywords and categories (6), an automatically generated, query-biased summary of the document content (7), a query-biased snippet based on the first hit within the document (8), and a term cloud showing the relative weight of the most important terms within the document (9). These pieces of information are geared towards allowing searchers to judge the relevance of a document without leaving the context of the search (while keeping the items in the result list to a moderate size that doesn't overwhelm with information). For quick fact-finding queries the query-biased summary might even allow a searcher access to the answer without consulting the full result document. Query terms are highlighted throughout the details.

This can be compared with the example of a result display for a scientific article (see Fig. 30. Here the system did not have access to the full text (and therefore couldn't provide an

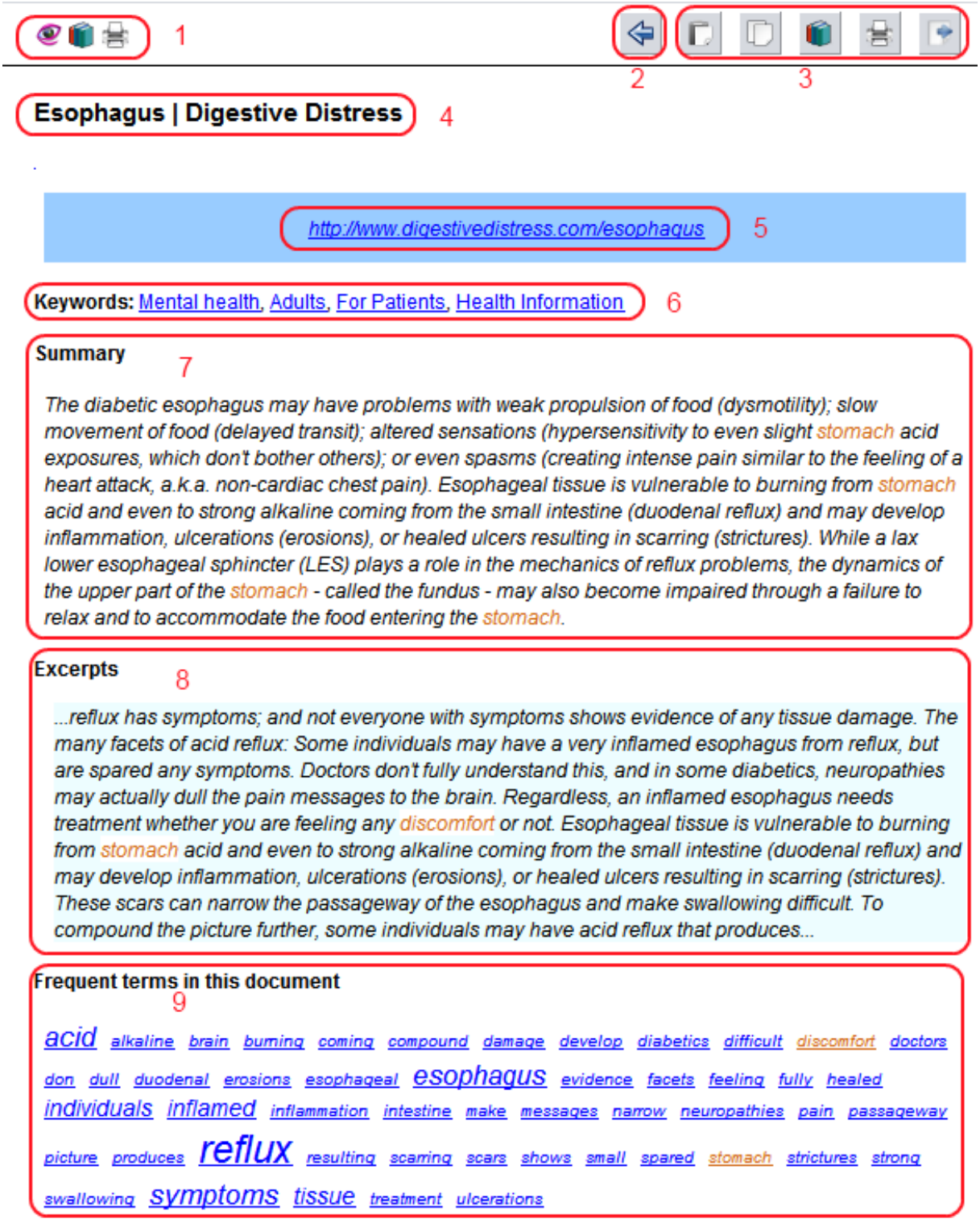

Figure 29: An example of result details for a web article.

automatically generated summary), but other types of meta data are available to help searchers determine the relevance of the document. Any part of the result displayed could be linked to

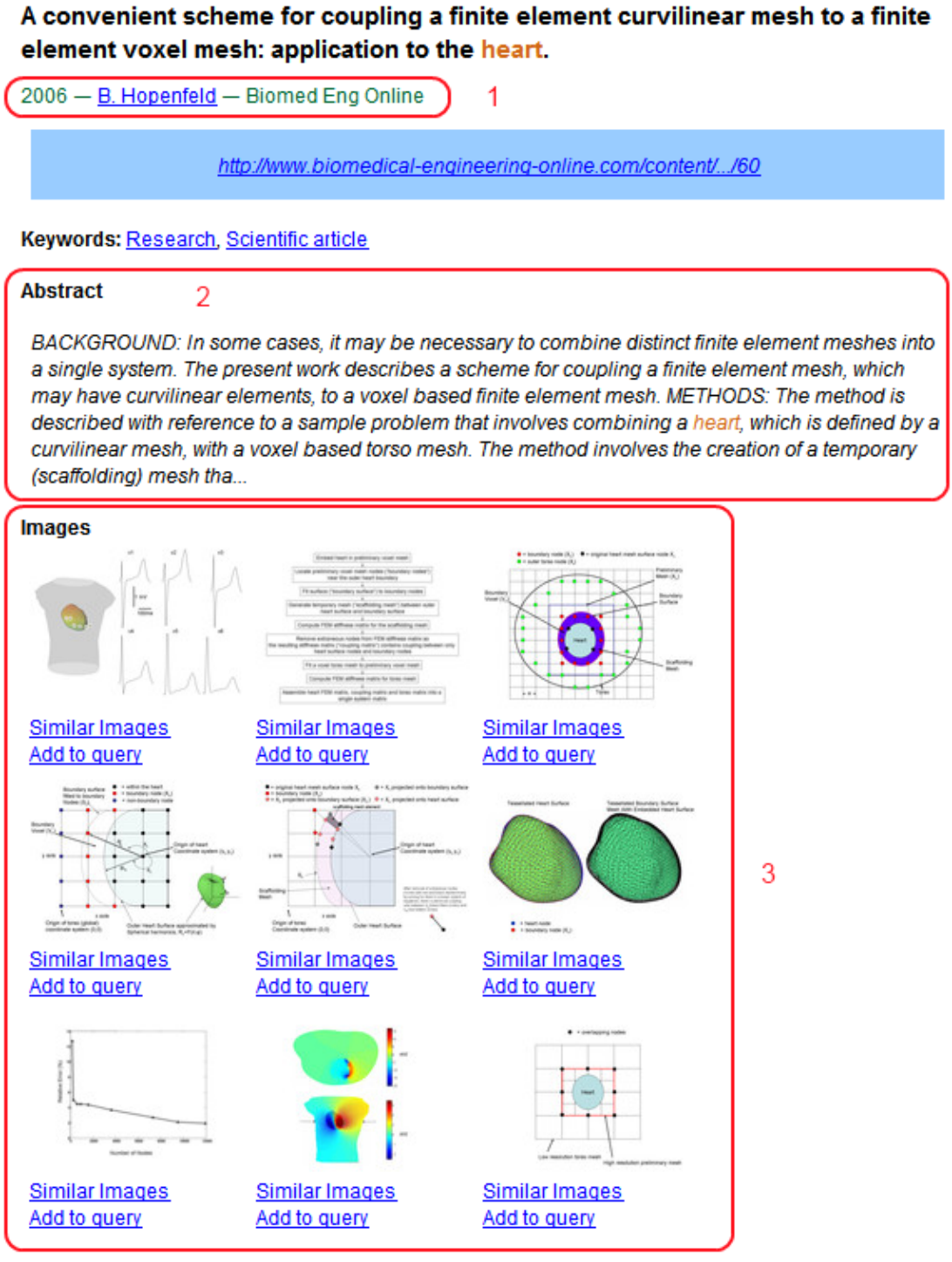

Figure 30: An example of result details for a scientific article.

new searchers or be used to restrict the current search by a filter criterion. In this example, authors, keywords and extracted images are provided as links.

For scientific articles, the publication date, authors, publication venue (1), and the abstract

of the article (2) are provided as additional information. Images from the article are shown with links to search for similar images like these (3). Clicking on the image thumbnail itself opens a full-size version. The full-size image can also be dragged to the query form to add it to a search for similar images.

Fig. 31 shows the result details for an image document. The original caption text of the image is shown with query terms highlighted. Links are provided to search for similar images or to add the image to an existing query specification, as well as to the original full-sized image and the document from which the image was extracted.

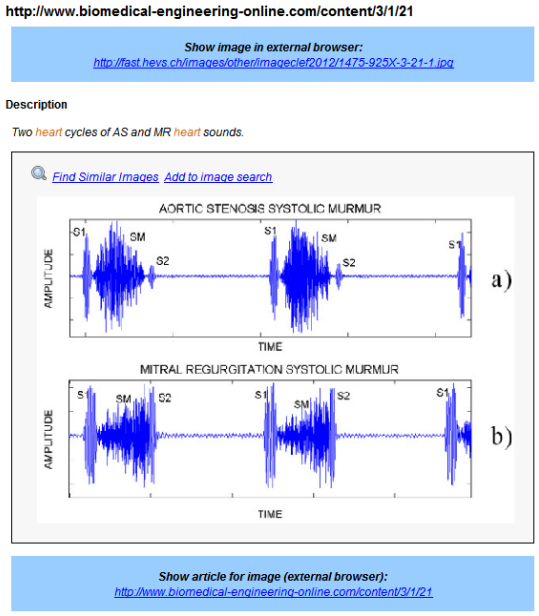

Figure 31: An example of result details for an image.

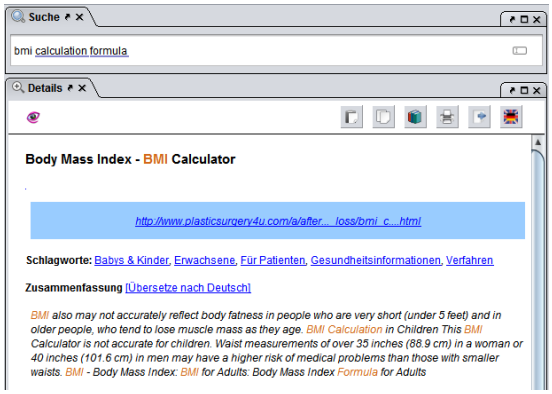

Figure 32: Translating parts of the result preview.

Since results can contain documents in different languages, the detail view also provides the possibility to translate the complete result details or parts of the details if they are not in the specified language of the user (see Fig. 32). As can be seen in the screenshot, text provided by the interface itself (such as field names or classification keywords) are always translated into the interface language (German in this case).

5 Personalization

Given the complexity of the full featured user interface and the large amounts of time long-term users such as physicians are believed to spend working with the system to satisfy complex information needs, the system should also support personalization. System personalization can reduce the amount of necessary actions or generally shorten task execution times. Commonly it is either possible to allow users the explicit manipulation of interface elements position or size or enable the system to automatically perform such changes based on user observation and task detection. The latter approach usually requires more design effort, e.g. creating a case base and deducting customization steps from that. The former gives users a subjective feeling of being in control over the system as it follows the principle of *controllability* (compare [11]). Therefore, the systems personalization mechanism is designed as described in the following sections.

5.1 Customization

The user interface provides several tools, each representing a set of related features. Tools can have a single non-modifiable view, e.g. the tray tool, or can have a number of views which may be adaptable, e.g. the query formulation tool. These tools are organized in perspectives which can be accessed via the menu bar. See section 2 for more details.

A user may choose to change a given perspective according to their needs by different means. On one side all tools and views may be opened and included in the current perspective or closed and excluded. Furthermore, all tools can be changed in size and moved around in the interface by means of drag and drop. It is also possible to undock a tool from the interface. Undocked tools are shown in a separate window which can be resized and positioned anywhere on the operating system's desktop.

All changes regarding the set-up of tools, the position in the perspective and the size and position of the views can be saved by the user. The configuration is serialized as an XML file, which makes it also possible to use a custom configuration on a different machine—assuming the display resolution is compatible.

The user preference menu can be used to select certain detail elements a user wants to include in the detail view if they are available for the result documents. For instance a user may decide to include a tag cloud view to gain a quick overview of terms that are common in a document. Others may prefer to see a web document only in its original format and therefore want to exclude the summary of a document or the translation option in the detail view.

Apart from explicit configuration options the interface also builds a user profile, which is described in the following chapter.

5.2 Personal profiles

The user interface framework features various means to log users' activities. Low-level logging on interaction level, like mouse clicks or actions performed is executed by *ezDL*. Furthermore, all search requests are logged. Users are able to give specific details to the system, e.g. demographic data. This information can be used to improve the user experience, e.g. by building perspectives derived from the user model. These perspectives many include tools that are likely to be of use to the type of user inferred from their profile. It is also possible to enrich the queries

with the user profile allowing the data sources to give back personalized results. The suggestion components introduced in Section 3.2.2 can also employ user profiles to improve suggestion quality.

The user model, defined in Khresmoi, consists of three differentiated models:

- User Profile: Defines the unique characteristics that distinctively identify a given user.
- Activity Model: Defines several activities that are generated and logged automatically, through user interaction with the system (see [4] for more details on the logging of events).
- Query Model: Describe the different types of queries and criteria supported by the system.

This personal profiling process will also focus on the identification of the main characteristics to describe the User Model. These characteristics include or could include the following:

- User Profile
- Identity (Name, Email, etc.)
- Preferences (Topics, language, etc.)
- Location (permanent, current)
- Health profile (age, height, weight, disease)
- Knowledge level (Novice, Intermediate, Expert, etc.)
- Kind of user (Patient, Doctor, Radiologist)
- Activity Model
- Query Definition (criteria selection, writing query, disambiguation, refinement, etc.)
- Results Activity (sorting selection, items selection, item bookmarking, item sharing, etc.)
- Content Activity (Item reading, item annotating)
- Query Model
- Query (Textual Query, Image Query, Multi-modal Query)
- Category, fields

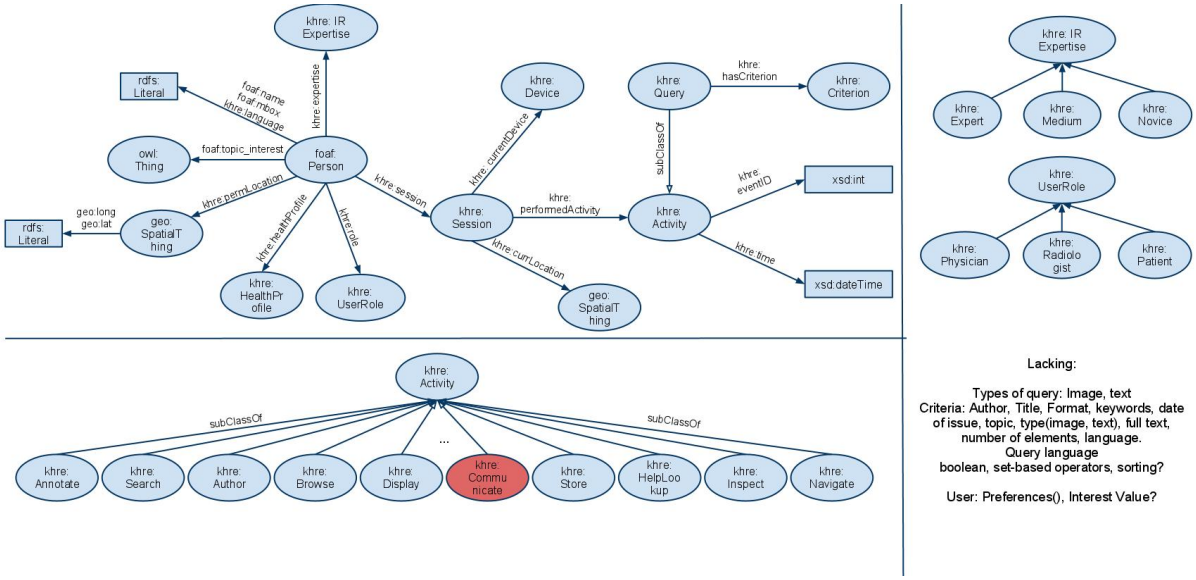

Figure 33: The RDF schema

5.3 Description of the RDF schema

Taking as reference the ontologies listed in the previous section with their main characteristics identified and defined for the User Model, a conceptual model was developed (compare Figure 33). The main elements of the schema are:

- Person: represents a system's user.
- Query: identifies a precise request for information retrieval within Khresmoi. There are two main types of queries: Visual and Textual.
- Session: identifies an atomic sequence of user interactions with the system, since they log in until they log out or are automatically logged out by the system. It is uniquely identified and it's comprised of the interaction characteristics, e.g. device used, connection (e.g. DSL, T1, modem, GSM) or user's location, and a sequence of user activities.
- Activity: identifies one specific kind of user interaction with the system. It's identified uniquely by an eventid and a timestamp. Each activity contains its own properties.
- ExpertiseLevel: represents the expertise of the user with this kind of information retrieval system.
- UserRole: specifies the professional user profile, either a general practitioner (Physician), a radiologist or a patient.
- HealthProfile: represents sensitive data regarding the user's own health. It must be treated with extreme care as explained below.
- Device: type and characteristics of the device used for the session.
- Connection: type and characteristics of connection for a session (e.g. up/down speed)

- Suggestion: represents the different analyses of the content added by the user in the system (query, annotations, etc.). Then, the system can suggest specific information to correct or disambiguate the data provide by the user, e.g. spelling suggestion, disambiguation, query refinement, etc.
- Recommendation: is a complementary dimension to the suggestion. Based on the user activities, the system should help the user to follow the most adapted situation. It should recommend some interesting search strategies according the current status of the search process (i.e. type or sequences of queries, types of results and topics often selected, types of document expected, user preferences, etc.).

5.4 Building user profiles from log data

The information gathered by *ezDL* will be stored in a triple repository. There are different solutions available:

- Sesame[1], an open standard framework for RDF data processing and storage. It provides inferencing and standard queries capabilities.
- Owlim[2], a semantic repository with support for more advanced semantics and better scalability and performance than Sesame.
- Otherwise, it uses a Sesame library to provide standard storage and querying APIs.

The architecture of the system is shown in Figure 34.

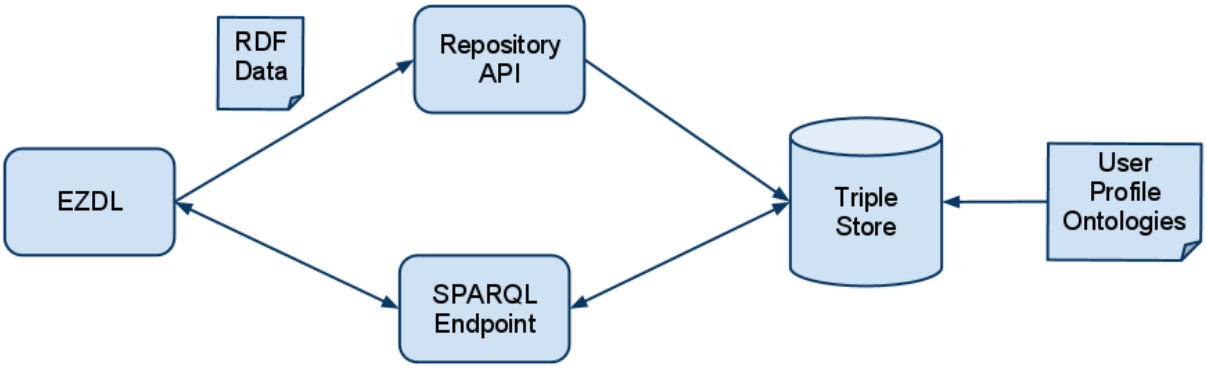

Figure 34: The log architecture.

EzDL will generate the data from the logs regarding the activities or events generated by the user in a session and transform them into RDF data according to the aforementioned ontologies. Afterwards, *ezDL* needs to consolidate the knowledge in the triple store. For that, it will need to use the repository API exposed by both Sesame and OWLIM, namely the SAIL API. During the retrieval phase, *ezDL* will construct and send queries to the exposed SPARQL endpoint via HTTP, using the SPARQL protocol.

[1] http://www.openrdf.org
[2] https://www.ontotext.com/owlim

It's very important that the architecture takes care of the privacy issues. There may be personal data that must be stored with the consent of the user, and needs to be highly secured according to European Data Protection regulations. In particular, it should abide by the European Data Protection Directive (Directive 95/46/EC), which regulates the processing and storage of personal data within the EU. This directive has been transposed into state member laws, and therefore, Khresmoi should follow that legislation as well. Specifically health data is considered sensitive and must be treated with the highest level of restrictions. These restrictions include that the data must be obtained only with the users' consent and that it can only be transferred to a third party with their consent, except for a few edge cases covered by national health legislation.

In regards to private health record information and following the current legislation, the users will be given the possibility to opt-in to introduce their health data.

6 Discussion and Conclusions

The user interface is a key component of every software system. This is also true for search systems in the medical domain. Interface design is crucial for a systems success. If it is well done, it enables user to interact with a system while concentrating on their actual work rather than on the interaction with the system itself [27]. To accomplish this goal, several design principles have been developed. The ones been relevant in the context of this system were elaborated in Section 2. Both Khresmoi user interfaces based on the *ezDL* infrastructure follow theses principles.

Nevertheless, it is a hard to decide at design or implementation time what tools a user might need for the task at hand. The one-size-fits-all approach is not applicable for the Khresmoi use cases as they differ too much in various dimensions. A variety of target groups require different features of the interface. One may be in need of extensive support, may it be with query formulation or search tactics in general, while another may do very well on its own.

The Khresmoi user interface framework is versatile enough to handle these situations. Query formulation for example can be performed in various ways with different interface elements. Suggestion agents deal with situations where user are might need help. Future developments in this area can be easily integrated in the interface through the plug-in system. Strategic support will improve over time, as the case base grows and user feedback improves suggestion quality. Result item presentation is designed to be flexible. E.g. site administrators can choose to customize the detail view and include or exclude certain elements form the view if it is needed for the use case. The system also features a option menu to allow customization by the users themselves. Furthermore, all perspectives can be adapted to a user's needs and stored for reuse.

While the system has been evaluated on a component level (compare [4]) it is yet to be evaluated in a larger scale user study. Such studies are currently underway and due to complete in April 2013 (D10.1 and D10.2). It is likely that further suggestions for improving the user interface will be made. This will result in a new versions of the interface components which will address possible shortcomings of the current ones. In addition, some of the components were finalized after the system was frozen for user testing, and their evaluation will be part of the second round of user evaluations.

References

[1] Marcia J. Bates. Idea tactics. *Journal of the American Society for Information Science*, 30(5):280–289, 1979.

[2] Marcia J. Bates. Information search tactics. *Journal of the American Society for Information Science*, 30(4):205–214, 1979.

[3] Marcia J. Bates. Where should the person stop and the information search interface start? *Information Processing and Management*, 26(5):575–591, 1990.

[4] Thomas Beckers, Sebastian Dungs, Norbert Fuhr, Lorraine Goeuriot, Jessica Ignalski, Matthias Jordan, Liadh Kelly, and Sascha Kriewel. Report on results of the WP3 first evaluation phase. Khresmoi Deliverable D3.2, 8 2012.

[5] Thomas Beckers, Sebastian Dungs, Norbert Fuhr, Matthias Jordan, and Sascha Kriewel. ezDL: An interactive search and evaluation system. In *SIGIR 2012 Workshop on Open Source Information Retrieval (OSIR 2012)*, August 2012.

[6] Nicholas J. Belkin. Anomalous states of knowledge as a basis for information retrieval. *Canadian Journal of Information Science*, 5:133–143, May 1980.

[7] Suresh K. Bhavnani, Christopher K. Bichakjian, Timothy M. Johnson, Roderick J. Little, Frederick A. Peck, Jennifer L. Schwartz, and Victor J. Strecher. Strategy hubs: Domain portals to help find comprehensive information. *Journal of the American Society for Information Science and Technology*, 57(1):4–24, 2006.

[8] Suresh K. Bhavnani, Bichakjian K. Christopher, Timothy M. Johnson, Roderick J. Little, Frederick A. Peck, Jennifer L. Schwartz, and Victor J. Strecher. Strategy hubs: next-generation domain portals with search procedures. In *Proceedings of the conference on Human factors in computing systems*, pages 393–400. ACM Press, 2003.

[9] Giorgio Brajnik, Stefano Mizzaro, and C. Tasso. Evaluating user interfaces to information retrieval systems: A case study on user support. In Hans-Peter Frei, Donna Harman, Peter Schäuble, and Ross Wilkinson, editors, *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 128–136, New York, 1996. ACM.

[10] Norbert Fuhr, Claus-Peter Klas, André Schaefer, and Peter Mutschke. Daffodil: An integrated desktop for supporting high-level search activities in federated digital libraries. In *Research and Advanced Technology for Digital Libraries. 6th European Conference, ECDL 2002*, pages 597–612, Heidelberg et al., 2002. Springer.

[11] Deutsches Institut für Normung. DIN En ISO 9241 Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten Teil 10: Grundsätze der Dialoggestaltung (ISO 9241-10), Deutsche Fassung, 1995.

[12] Lorraine Goeuriot, Allan Hanbury, Gareth Jones, Liadh Kelly, Sascha Kriewel, Ivan Martinez Rodriguez, Henning Müller, and Miguel A. Tinte. Supporting collaborative improvement of resources in the khresmoi health information system. In *LREC*, 2012.

[13] Jessica Ignalski, Matthias Jordan, and Sascha Kriewel. Evaluierung von Darstellungsvarianten für Anfragevorschläge bei der Informationssuche. In *Proceedings of the IR Workshop at LWA 2012, Dortmund, Germany*, September 2012.

[14] Hideo Joho and Joemon M. Jose. Effectiveness of additional representations for the search result presentation on the web. *Information Processing & Management*, 44(1):226 – 241, 2008.

[15] Matthias Jordan. DAFFODIL: Proaktive Vorlagefunktionen. Diplomarbeit, Universität Dortmund, FB Informatik, 2005.

[16] Claus-Peter Klas, Sascha Kriewel, and André Schaefer. Daffodil - Nutzerorientiertes Zugangssystem für heterogene digitale Bibliotheken. *dvs Band*, 2004.

[17] Sascha Kriewel. *Unterstützung beim Finden und Durchführen von Suchstrategien in Digitalen Bibliotheken*. PhD thesis, University of Duisburg-Essen, 2010.

[18] Sascha Kriewel. Interaktives Retrieval und situationsabhängige Vorschläge. *Datenbank-Spektrum*, 11, 2011.

[19] Sascha Kriewel and Norbert Fuhr. Adaptive search suggestions for digital libraries. In *Asian Digital Libraries: Looking Back 10 Years and Forging New Frontiers (ICADL 2007)*, pages 220–229, 2007.

[20] Sascha Kriewel and Norbert Fuhr. An evaluation of an adaptive search suggestion system. In *32nd European Conference on Information Retrieval Research (ECIR 2010)*, pages 544–555. Springer, 2010.

[21] Sascha Kriewel, Claus-Peter Klas, André Schaefer, and Norbert Fuhr. Daffodil - Strategic support for user-oriented access to heterogeneous digital libraries. *D-Lib Magazine*, 10(6), June 2004.

[22] Gary Marchionini. Information-seeking strategies of novices using a full-text electronic encyclopedia. *Journal of the American Society for Information Science*, 40(1):54–66, 1989.

[23] Melikka Khosh Niat and Norbert Fuhr. Are ontologies helpful in medical search? In *Proceedings of the IR Workshop at LWA 2012, Dortmund, Germany*, September 2012.

[24] Konstantin Pentchev. Large scale biomedical knowledge server. Khresmoi Deliverable D5.2, 8 2012.

[25] André Schaefer, Matthias Jordan, Claus-Peter Klas, and Norbert Fuhr. Active support for query formulation in virtual digital libraries: A case study with DAFFODIL. In A. Rauber,

C. Christodoulakis, and A M. Tjoa, editors, *Research and Advanced Technology for Digital Libraries. Proc. European Conference on Digital Libraries (ECDL 2005)*, Lecture Notes in Computer Science, Heidelberg et al., 2005. Springer.

[26] Ralf Schlosser, Oliver Wendt, Suresh Bhavnani, and Barbara NailChiwetalu. Use of information-seeking strategies for developing systematic reviews and engaging in evidence-based practice: the application of traditional and comprehensive Pearl Growing. a review. *International Journal of Language & Communication Disorders*, 41:567–582(16), September-October 2006.

[27] Ben Shneiderman. *Designing the user interface: strategies for effective human-computer interaction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[28] Andreas Tacke. Integration von strategischer Suchunterstützung auf Makro- und Mikroebene. Master's thesis, University of Duisburg-Essen, Information Engineering, 2013.

[29] Jenifer Tidwell. *Designing Interfaces: Patterns for Effective Interaction Design*. O'Reilly Media, Inc., November 2005.

[30] Fadhilah M. Yamin and T. Ramayah. User web search behavior on query formulation. In *Semantic Technology and Information Retrieval (STAIR), 2011 International Conference on*, pages 182 –188, june 2011.

A The ANTLR grammar

```
// ANTLR 3 grammar
//
// Possible queries are like these:
// term
// field=term
// field>=term
// (field=term AND otherfield<something) AND test OR <5
//
grammar WebLike;

options {output=AST;}


@parser::header
 { package de.unidue.inf.is.ezdl.dlfrontend.query.parsers.simple; }
@lexer::header
{ package de.unidue.inf.is.ezdl.dlfrontend.query.parsers.simple; }

// Make sure that unrecognized tokens (e.g. unclosed phrases)
// are not ignored but lead
// to a proper RecognitionException.
@lexer::members{ public void recover(RecognitionException re)
{ throw new RuntimeException(re); } }



query : (WS!)? subquery (WS!)? EOF;


subquery : andclause;


pseudoatomic: fieldexpression
            | PAROP! (WS!)? subquery (WS!)? PARCLS!
            | proximityclause;


////////////////////////////////////////////////////////////////////
//
// "Normal" operator priority
// Here, this means OR before AND, to make it easier to specify synonyms:
// syn1 OR syn2 OR syn3 AND term2 means (syn1 OR syn2 OR syn3) AND term2
//
```

```
// 'AND'ing something either so "a b c" or so "a AND b AND c".
//
andclause: (a=orclause->$a) ( ( impland b+=orclause )+ -> ^(AND $a $b+) )? ;

fragment impland: WS | (WS!)? AND (WS!)?;

fragment or: (WS!)? OR (WS!)?;


orclause: (a=pseudoatomic->$a) (( or b+=pseudoatomic )+ -> ^(OR $a $b+) )?;



////////////////////////////////////////////////////////////////////////
//
// Proximity search
// Applies only to raw terms or phrases. Not to fieldterms to prevent
// Author=A NEAR/5 Title=B from being legal.
//

proximityclause:
a=phraseterm WS NEAR? SLASH NUMBER WS b=phraseterm -> ^(SLASH $a NUMBER $b);


////////////////////////////////////////////////////////////////////////
//
// Field braces. Sets the default field for the subquery.
//
// To specify stuff like Title={a OR b AND c} which is way nicer than
// Title=a OR Title=b AND Title=c
//

fieldbrace:
f=fielditentifier EQL BRACEOP WS? s=subquery WS? BRACECLS -> ^(BRACEOP $f $s);



////////////////////////////////////////////////////////////////////////
//
// Only simple term-related stuff below this line
//

fieldexpression: fieldterm
                | NOT WS fieldterm -> ^(NOT fieldterm)
```

```
                | NOTm fieldterm -> ^(NOT fieldterm);


fieldterm: phraseterm  // term
          | compop^ phraseterm // >=term
          | fieldidentifier compop^ phraseterm // field>=term
          | fieldbrace; // Field={...}

fragment compop: EQL|GT|LT|GTE|LTE;

fragment fieldidentifier: TEXT;



phraseterm: phrase | wildcardedterm;

phrase: PHRASE;

wildcardedterm: (a=term->$a) ((b+=wildcardterm)+ -> ^(PLACE_CHAR $a $b+))?;

fragment wildcardterm: wildcard | term;
fragment wildcard: PLACE_CHAR | PLACE_CHARS;

term : TEXT | NUMBER;


////////////////////////////////////////////////////////////////////////
//
// Lexer rules -- what do the atoms look like?
//

// If the place holder characters are changed, be sure to update TEXTHEAD.
PLACE_CHAR : '$';
PLACE_CHARS : '#';

PHRASE: '"' .* '"';

PAROP: '(';

PARCLS: ')';

BRACEOP: '{';

BRACECLS: '}';
```

```
AND: 'AND';

OR: 'OR';

NEAR: 'NEAR';

SLASH: '/';

NUMBER: '0'..'9';

NOT: 'NOT';
NOTm: '-';

EQL: '='|':';

GT: '>';

LT: '<';

GTE: '>=';

LTE: '<=';


// See http://www.unicode.org/charts/PDF/U0000.pdf
// for a map between ASCII and unicode.

fragment TEXTHEAD:
'!'|'%'..'\''|'*'..','|'.'|'0'..'9'|';'|'?'..'z'|'|'|'~'|'\u00a0'..'\udfff';
TEXT: TEXTHEAD (TEXTHEAD|NOTm)*;

WS: (' '|'\r'|'\n')+;
//{$channel = HIDDEN;};
```