Grant Agreement Number: 257528

KHRESMOI

www.khresmoi.eu

D3.6: Final flexible user interface framework and documentation

Deliverable number	*D3.6*
Dissemination level	*Public*
Delivery data	*28 February 2014*
Status	*Final*
Authors	*Thomas Beckers, Sebastian Dungs, Norbert Fuhr, Matthias Jordan, Sascha Kriewel*

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 report describes the flexible user interface framework developed as part of task 3.1 in work package 3 (“User Interface and Search Specification System”) of the Khresmoi project. It provides three frontends with an open set of tools which group logically related user interface functionalities. Tools are loosely coupled, can be configured into task-based perspectives and invoked through drag and drop interaction. Users can personalize the system through configuration of new perspectives, filters, result presentations, preferred query language and detail aspects shown.

The backend of the framework is composed of independent agents, each either providing specific system services such as user authentification, logging, meta search, caching, or document storage - or connecting to web services that offer additional functionalities. The web services provided by the integration layer of the Khresmoi project (e.g. translation, spelling correction, disambiguation, semantic lookups, annotation storage) are connected to the Khresmoi Professional interface prototype by individual backend agents.

Table of Contents

Abbreviations

API	Application Programming Interface
BibTeX	Bibliographic system for TeX and LaTeX
DAFFODIL	Distributed Agents for User-Friendly Access of Digital Libraries
CORBA	Common Object Request Broker Architecture
DCCP	Dynamic Client Configuration Protocol
DL	Digital Library
DOI	Digital Object Identifier
EZDL	easy access to Digital Libraries
GWT	Google Web Toolkit
ISBN	International Standard Book Number
JMS	Java Message Service
JSON	JavaScript Object Notation
KMI	Khresmoi Mímir Interface
KP	Khresmoi Professional
MTA	Message Transfer Agent
OID	Object IDentifier
OS	Operating System
ParaDISE	Parallel Distributed Image Search Engine
RDF	Resource Description Framework
REST	REpresentational State Transfer
RIA	Rich Internet Application
RIS	Reference Information Systems
SCA	Service Component Architecture
SOAP	Simple Object Access Protocol
URI	Unique Ressource Identifier
URL	Unique Ressource Locator

1 Introduction

This deliverable reports on the prototype of the flexible user interface framework that is used in the Khresmoi Professional search system. The framework developed in work package (WP) 3 is based on the EZDL (easy access to Digital Libraries) architecture described in [6, 7]. It uses an extensible and service-oriented architecture where various backend functionalities are implemented as independent software agents. User clients for the desktop, web browsers and mobile devices have been developed to connect to the common desktop architecture. EZDL is the successor of the Daffodil (Distributed Agents for User-Friendly Access of Digital Libraries) system, a metasearch system for searching in heterogeneous, distributed digital libraries [11, 20]. In addition to combining access to multiple libraries and other web services, the system also incorporates cognitive models of information seeking and searching, as well as proactive search assistance [24] and support for collaboration. These concepts behind the development of the framework are discussed in Section 2 of this document.

The framework is divided into the backend, which is composed of multiple agents, and a number of different frontend implementations. The backend is detailed in Section 3, while Section 4 describes the different frontends for the Khresmoi Professional interface framework. In Section 5 the installation process and the customization of an installation are described. A full installation guide can be found in [16].

2 General overview and concepts

The developed framework uses the EZDL system as its base. EZDL is a system for building interactive search applications and is the successor of the Daffodil system that was developed in the early 2000s. EZDL supports the user not only during the actual search actions but also afterwards, e.g. while organizing found documents. The system is designed to connect to various document repositories or digital libraries. Thus, EZDL allows the user to perform meta-searches in different document collections at the same time with an integrated search system.

The EZDL system can be configured and extended for building custom search applications. This was done for the Khresmoi project to develop a search user interface framework tailored to the requirements of medical professionals.

Finally, the framework offers support for performing user studies. All user and system actions are logged for later analysis. Furthermore, there is a special evaluation mode that can disable unused interface elements or show search tasks to the user.

2.1 Major Concepts

EZDL and by extension the Khresmoi interface framework builds upon various concepts for supporting users with complex work tasks. The following subsections give a short overview of the major concepts.

2.1.1 Tactical search support

Bates describes different levels of search activities: *moves*, *tactics* and *stratagems* [1, 2]. Building upon basic activities (called *moves*) *tactics* and *stratagems* are higher-level activities to advance a search. Some of these are currently supported by EZDL, e.g. term suggestions for the search query or the extraction of frequent authors or terms from the result list. Suggestions of *tactics* and *stratagems* for furthering the search are shown to the user on request based on case-based reasoning and previous feedback from other users about the usefulness of the suggestions [18, 19].

2.1.2 Digital Library Lifecycle

The *Digital library Lifecycle* was proposed by Paepcke [23]. It comprises the different stages of the information work-flow during searching. The first stage is the *discovery* of information sources. The next stage is the *retrieval* of relevant information. After the actual searching the information is *collated*, e.g. by tagging objects or by storing found objects in a personal library. During the *interpretation* stage the user interprets the found information. Finally the lifecycle concludes with *representing* new information. Each stage is supported by EZDL (see Section 4.1 for more a detailed explanation of the various tools).

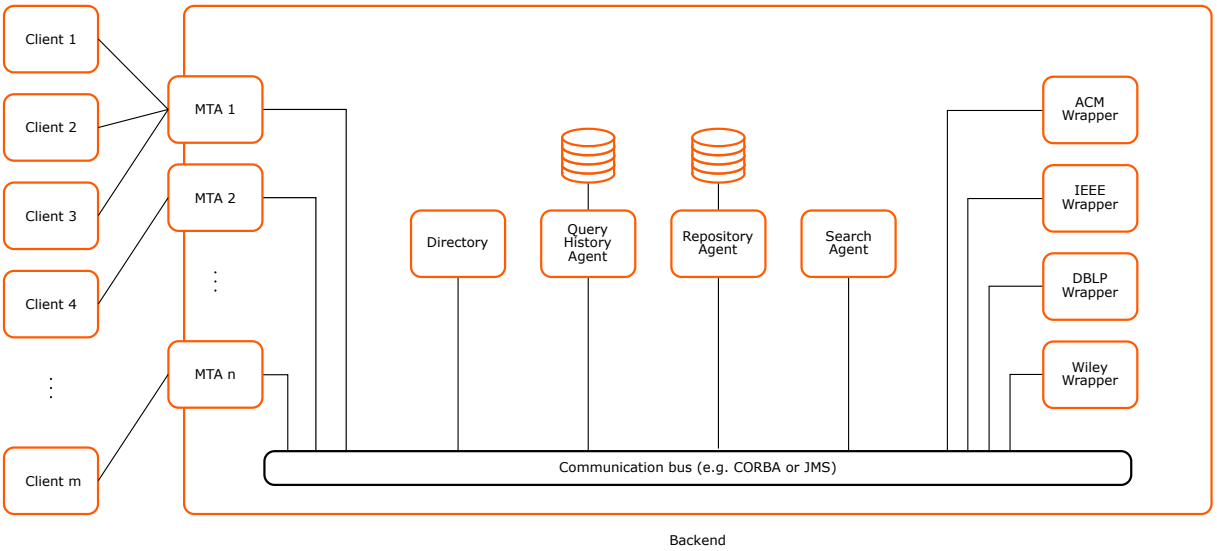

Figure 1: The architecture of the backend .

2.1.3 Collaboration

EZDL supports collaboration between users during their information seeking process. Users can share documents in the personal library with other users but also with user groups. Another collaborative feature of EZDL is the possibility for users to annotate documents. An annotation is a short text created by the user that is attached to a document. Annotations can be *public* or *private*. Private annotations are only visible for the annotation owner. Public annotations are visible for the public.

2.1.4 Proactivity

Bates proposed five levels of system involvement [4]. On the higher levels actions are performed by the system without asking the user in advance (proactively). The proactive functionality for query suggestions in EZDL belongs to level 3. That means that the system suggests corrected or related terms automatically during the query formulation. The system observes the user input while formulating a query. Based on the query terms and operators observed suitable term suggestions are fetched and shown to the user.

2.2 System Architecture

The EZDL system comprises a *backend* and various *frontends*. The main client for Khresmoi Professional is the desktop frontend. Other clients include an Android-based mobile application [22] and the browser frontend. Beckers et al. [6] provide a more detailed overview of the system architecture.

The architecture of the backend is shown in Fig. 1. The various frontends are shown on the left-hand side. On the right-hand side is the agent-based backend that will be described in more detail in the following section. Message Transfer Agents (MTAs) provide the connection between the client and the backend layer (see Section 3.1.2).

The desktop client architecture (see Fig. 2) uses loosely-coupled *tools* for the different search and seeking features.

3 Backend and agent framework

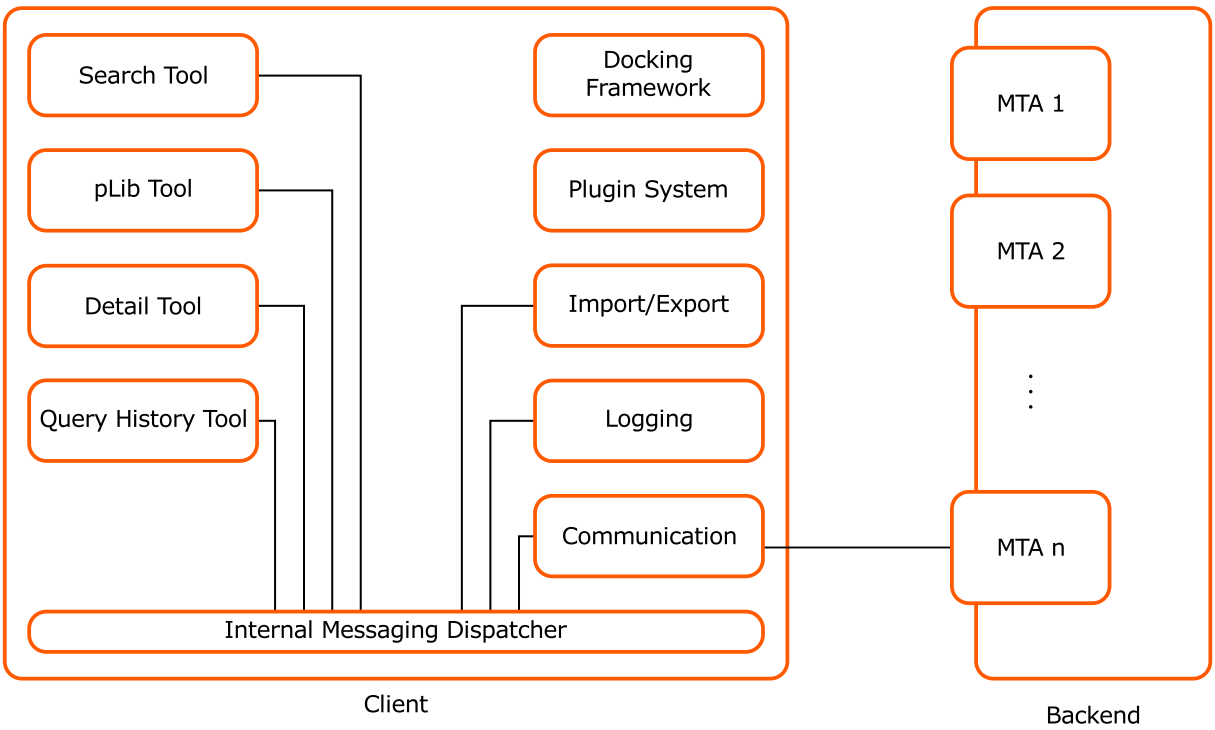

Figure 2: The architecture of the desktop client

3.1 Agent framework

The backend consists of a collection of agents, independent processes specialized for a specific task and handling a set of messages.

The agents communicate with each other using a shared communication infrastructure. While it is possible to use a wide variety of infrastructures [1], the current implementation uses Java Message Service (JMS). This section describes the most important agents of the user interface framework, which provide the central services of user authentication, communication, message transfer, search and caching.

The messages sent between parties in the backend follow a simple naming scheme: Messages whose names end with "Ask" are requests that are expected to be answered by a reply message. Messages whose names end with "Tell" are answers to Ask messages. A third group is messages whose names end with "Notify". These messages are sent to a receiver without expecting a reply. Typical examples for Ask messages are search queries. These queries would be answered by a Tell message containing a result list. A typical Notify message is the message that an agent sends to the directory when going offline.

3.1.1 The directory agent

The Directory Agent fills a central role in the EZDL backend and is the first agent that should be started. It can be used by other agents to find out which agents provide a specific service (e.g., Search or Translation). The directory agent also provides an administration interface to the site admininstrator, which allows operators to query information about all known agents or to stop them. The information available is the list of requests that are currently being processed, the status of the agent (including how many threads are running), how many errors happened in a configurable time frame and a log of the last things that the agent did. See Fig. 3 for a screenshot of the directory web page.

The directory agent handles the following messages:

[1] An earlier version of EZDL used CORBA

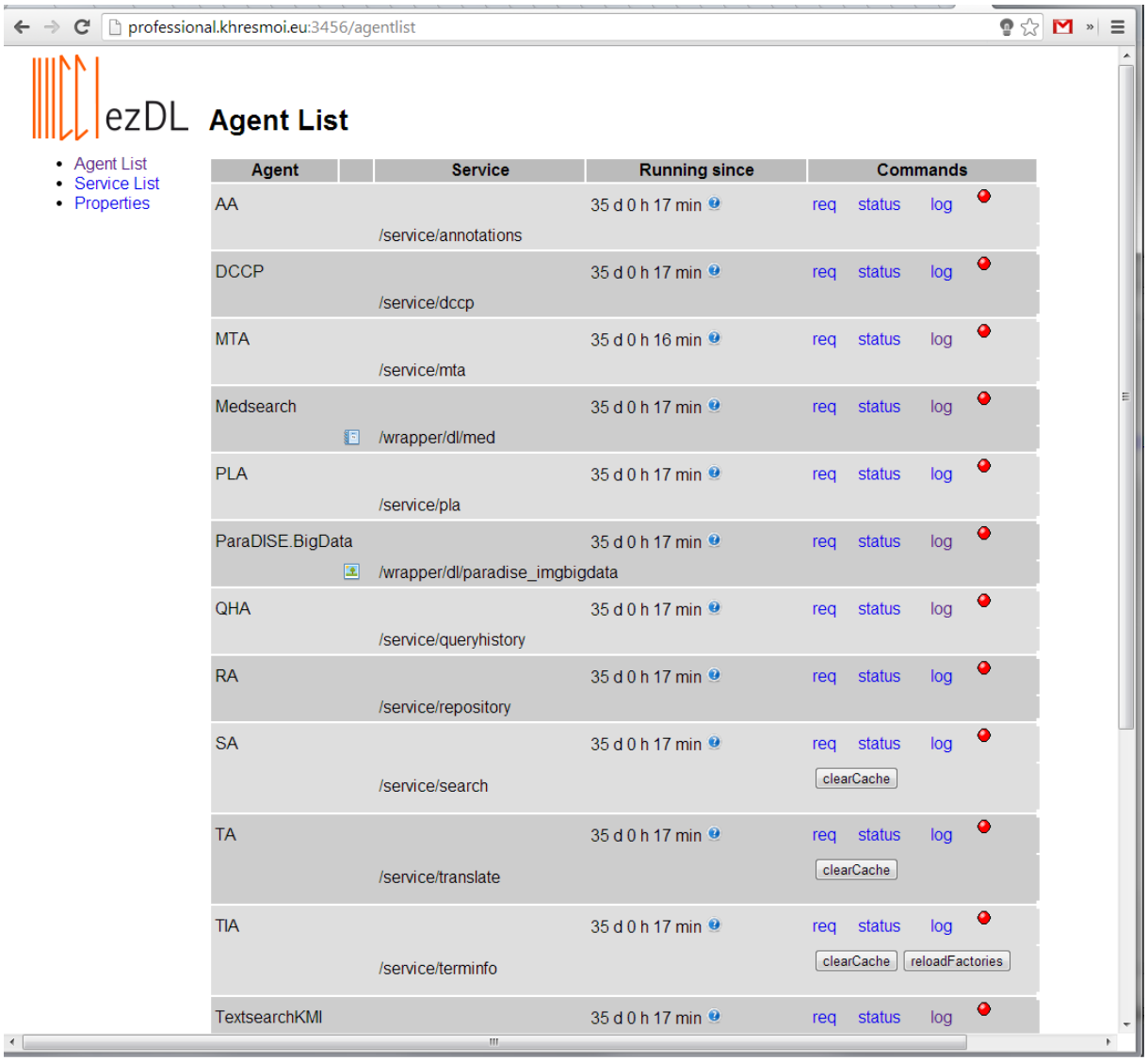

Figure 3: Administrative web interface for agents.

- AgentNameAllAsk: Asks for the names and registration information of all agents that implement a service given by its descriptor (e.g. `/wrapper/dl/acm`).
- AgentNameAsk: Asks for the name and registration information of any agent that implements a given service.
- AliveTell: The answer to an `AliveAsk` informs the directory agent that the sender is still alive.
- RegisterAsk: Asks the directory to register the sender, submitting information about the services the sender implements.
- DeregisterNotify: Notifies the directory that the sender is about to quit and that the directory should remove its data from the internal agent list.
- AvailableWrappersAsk: Asks the directory for information on available wrappers and submits the locale that the answer is to be returned in.

3.1.2 Message Transfer Agent

The interface for external clients of the backend is provided by a special class of agents, the Message Transfer Agents (MTA). An MTA implements an external protocol (e.g. the Simple Object Access Protocol or SOAP) and provides certain services to clients of this protocol by translating the clients' requests to messages for other agents in the backend. It routes messages to other agents using a routing table, and also pushes notification messages to an online client associated with a specific user.

In addition to routing all backend messages, the Message Transfer Agent handles these messages itself:

- PushMessageToUserNotify: Notifies the MTA that an enclosed push message should be forwarded to the connection with the given session ID.
- UserMessageNotify: Notifies the MTA that an enclosed text message is to be forwarded to either a specific session ID or broadcast to all sessions that the MTA is handling.

3.1.3 The search agent and the search subsystem

Search is handled by the search agent. When a client sends a search request to the search agent, the search agent broadcasts the request to certain agents that encapsulate actual document collections, the wrapper agents.

Each wrapper agent deals with a specific collection (e.g. the ParaDISE[2] big data image collection, or an internal Solr server). It translates internal queries into those understood by the document collection and representations of the document collection back into internal document representations. When a wrapper agent receives a search query from a client, it communicates the query with the document collection and returns matching documents to the client who sent the search query – in most cases the search agent.

The search agent collects the answers from all contacted wrapper agents and merges them into one result list. After the results are complete, the items can be re-ranked (or the original ranking of the source can be kept with the results from different sources being interleaved). The parts of the documents that the client is interested in, (e.g. title, authors, abstract) are then returned to the client.

Also, a copy of the full results (all parts) is sent to the repository agent so it can collect information about known objects.

The messages handled by the search agent are:

- DLObjectQueryAsk: Asks the search agent to search for documents matching the enclosed query and return documents that fit the enclosed description (e.g. result item position range, set of requested fields and sorting).
- CancelSearchNotify: Notifies the search agent that a given query is no longer needed to be processed. It can be used to request partial results to be returned immediately.

3.1.4 The repository agent

The repository agent keeps information about all known objects. Clients can ask the repository agent for information on an object based on its internal ID.

The repository agent is responsible for handling these messages:

- DLObjectDetailsAsk: Asks the repository to return details about objects given by their object IDs.
- DLObjectDetailsFillTell: The answer of a wrapper to a request of the repository to find new data about objects.
- AddToRepositoryNotify: Notifies the repository agent about a new object to be stored.
- AddNewIDNotify: Notifies the repository that new IDs have been assigned to some objects.
- DLObjectQueryStoredTell: The "carbon copy" of a result set that the search agent found. The result set is also sent to the repository so the repository knows about the objects and the wrapper that found them. The repository can then later ask the originating wrapper for details.

[2] Parallel Distributed Image Search Engine

3.1.5 The user agent

All information about users is kept by the user agent. The user agent offers a number of different services related to registered users: manipulating user information, managing user groups, authentication and authorization, and searching for users. Other agents that need to check authorization for specific tasks, such as storing documents in a personal collection, request this information from the user agent. Authorization is role based: users can have multiple roles, and each role can be given a set of privileges. The messages handled by this agent are:

- LoginAsk: Asks to login the user using a username and password.
- PrivilegeAsk: Asks for a set of privileges for the given user.
- StoreConfigNotify: Notifies the agent about a change to the user's configuration that should be stored in the backend.
- GroupCreateAsk: Asks to create a group. The user agent replies with the generated ID of the new group.
- GroupDeleteNotify: Tells the user agent to delete a group, if the user is allowed to.
- GroupInfoAllAsk: Requests information about all groups the sender is allowed to see.
- GroupInfoAsk: Requests information about a specific group.
- GroupInfoStoreAllNotify: Tells the user agent to store information about groups.
- GroupListAsk: Asks for the list of groups that the sender is allowed to see.
- GroupAddUsersAsk: Adds users to a group. The user agent replies with updated information about the group.
- GroupRemoveUsersAsk: Removes users from a group. The user agent replies with updated information about the group.
- PublishPushMessageNotify: Tells the user agent to deliver a push message to a user.
- PushMessageConfirmTell: Tells the user agent that a push message has been confirmed by the receiver.
- StoreUserNotify: Tells the user agent to store data about a user.
- UserListAsk: Asks for a list of users that share a given prefix.
- UserIdAsk: Asks for the ID of the user who is assigned to a given session ID.
- UserListByIdAsk: Asks for a list of users that correspond to a set of user IDs.
- UserListByLoginAsk: Asks for a list of users that correspond to a set of logins.
- DLObjectQueryAsk: The request for users that match a given query.
- DLObjectDetailsFillAsk: The request for detailed information concerning given users.
- StoreConfigNotify: Notifies the user agent about configuration items to be stored for later retrieval.

3.2 Additional agents

The following section describes the additional available agents for the Khresmoi Professional backend and documents the messages supported. While the agents described in the previous section provide central services that are necessary for the functioning of the Khresmoi Professional backend, these agents implement optional services.

3.2.1 Annotation Agent

The Annotation Agent handles storing and fetching of annotations on documents. The annotation store is provided by the Correction Manager component, a RESTful webservice described in detail in deliverable D3.4 [13]. The messages that are handled by this agent are:

- AddAnnotationAsk: Asks the agent to store a new (public or private) annotation for an object in the specified language.
- GetObjectAnnotationsAsk: Asks the agent to fetch all visible annotations for an object (optionally translated into the specified language).

- GetUserAnnotationsAsk: Asks the agent to fetch all visible annotations by a user (optionally translated into the specified language).
- GetAnnotationByIdAsk: Asks the agent to fetch an individual annotation (optionally translated into the specified language).
- UpdateAnnotationAsk: Asks the agent to update the content of a given annotation.
- DeleteAnnotationAsk: Asks the agent to delete a given annotation.
- AddVoteAsk: Asks the agent to add a positive or negative vote to an object (such as a document, annotation or query).
- RemoveVoteAsk: Asks the agent to remove a vote from an object.
- GetVoteAsk: Asks the agent to fetch all votes for a given object.

3.2.2 Dynamic Client Configuration Protocol (DCCP) Agent

The DCCP Agent provides clients with configuration information about the backend. Instead of directly connecting to a known Message Transfer Agent, clients can request a property file for a given protocol with the necessary connection information for this protocol. Multiple DCCP servers for different protocols may be used, e.g., one for external clients that want to use an MTA and one for clients that want to use a SOAP server.

3.2.3 Log Agent

The Log Agent handles the storing of log events. The access is hidden behind an abstract store and can be implemented by a relational database system, an RDF[3] tuple store (such as the one described for building user profiles in D3.3 [9]), or even a set of flat files. The agent handles only a single message (UserLogNotify), which describes a single log event with an event type, a session id, an event sequence, a timestamp and a key-value map of event parameters (such as query terms for a search, or the URL of a detail viewed by the user).

3.2.4 Personal Library Agent

The Personal Library Agent handles storing, fetching, sharing and tagging of objects in the personal collection of registered users. The access is hidden behind an abstract store. The current implementation of the personal library uses a relational database system, but the actual implementation can easily be changed. The messages that are handled by this agent are:

- StorePLibEntriesAsk: Asks the agent to add a new document, query or other object to the personal collection of the current user.
- DeletePLibEntriesNotify: Asks the agent to delete a specified object from the personal collection of the current user.
- PLibEntriesAsk: Asks the agent to fetch all entries from the current user's personal collection.
- SharePLibEntriesAsk: Asks the agent to share a set of objects from the current user's collection with a set of other users.
- UnsharePLibEntriesAsk: Asks the agent to unshare a set of objects from the current user's collection.
- AcceptSharePLibEntriesNotify: Notifies the agent of accepting or declining the offered share of one or more documents.
- AddTagsToPLibEntriesNotify: Notifies the agent that a set of tags (personal, descriptive labels) has been added to a set of objects from the current user's collection.
- DeletePLibTagsNotify: Notifies the agent that a set of tags has been deleted from an object in the current user's collection.

[3] Resource Description Framework

3.2.5 Query History Agent

The Query History Agent handles storing and fetching of search history items for users with a personal account. The access is hidden behind an abstract store and can be implemented using a relational database system, an RDF tuple store or even flat files. The messages that are handled by this agent are:

- StoreQueryHistoryNotify: Notifies the agent that a single search history item should be stored for the current user.
- ClearQueryHistoryNotify: Notifies the agent that the complete search history of the current user has been cleared.
- DeleteQueryHistoryNotify: Notifies the agent that a specific search history item of the current user has been deleted.
- QueryHistoryAsk: Asks the agent to fetch a given number of search history items starting with the N-th item.

3.2.6 Semantic Agent

The Semantic Agent handles fetching of semantic term suggestions and fetching of images based on semantic information about modalities, pathologies and anatomies. The messages that are handled by this agent are:

- GetSemanticTermsAsk: Asks the agent to fetch common anatomies, pathologies and modalities to provide as image search suggestions
- SemanticImageSearchAsk: Asks the agent to fetch images and associated terms for given anatomies, pathologies and modalities

3.2.7 Suggestion Agent

The Suggestion Agent handles the extraction of case information from user actions necessary for the case-based reasoning used to provide search continuation suggestions (as described in deliverable D3.3 [9]). It also fetches these suggestions based on the current user situation, and allows storing of user feedback for suggestions. The messages that are handled by this agent are:

- DLObjectQueryAsk: This message to the search agent is copied for the suggestion agent on a newly started search. A new case is created and filled with context data from the query itself.
- DLObjectQueryResultTell: When a search result returns, the case related to the original search request is updated with context data extracted from the result.
- GetSuggestionsAsk: Asks the agent to fetch the most similar cases for the current search situation and return a ranked list of suggestions associated with those cases.
- SuggestionFeedbackNotify: Notifies the agent of negative or positive feedback information for a given case.

3.2.8 Terminfo Agent

The term information agent is the resource for (linguistic) information about terms. It can be asked for several types of information (e.g. synonyms or spelling corrections or both) and returns the desired information to the client. The agent uses an extensible set of modules to ask internal or external data sources. Examples for external data sources are the Ontotext disambiguation service, which is contacted via a component of the Service Component Architecture (SCA) in the integration layer, and the spelling service provided by another integration layer component.

- TermsInformationAsk: Asks the agent to fetch a ranked list of suggestions for a given term; the suggestions can be of the types RELATED (including disambiguation suggestions), SYNONYM (alternate names), RESPELL (spelling corrections).
- EntityDetailsAsk: Asks the agent to fetch additional information for a given semantic concept, such as definitions or alternate labels.
- RelatedArticleAsk: Asks the agent to fetch the associated article for a given image.

3.2.9 Translation Agent

The Translation Agent handles fetching and updating of document and query translations. The translations (for the language pairs English/German, English/French and English/Czech) are provided by the service described in deliverable D3.1 [12]. The messages that are handled by this agent are:

- TranslatedDocumentAsk: Asks the agent to translate the complete details of a found document (title, summary, abstract and/or snippets).
- TranslatedQueryAsk: Asks the agent to translate a query or a single query term.
- TranslatedTextAsk: Asks the agent to translate a longer text with alignment info.
- UpdateTranslationNotify: Notifies the agent of an update for a translation.

3.3 Wrappers

Wrappers are specialized agents connecting a specific document collection (such as a digital library, search engine, or a local index). These agents translate the internal query tree representation of user search requests into queries that can be resolved by the remote source, and convert results into the internal document representations used by the interface framework.

- DLObjectQueryAsk: Asks the search system to query a set of remote document collections using a user query, a filter condition, context information, a list of wrappers to ask, and a result configuration. Through the result configuration the client can request a specific chunk of a result set (e.g., results N to N+50) or specify which fields it is interested in (to reduce the size of the initial result set, only those fields necessary to display a result list are requested).
- DLObjectDetailsFillAsk: Asks the search system to fill the missing details for a document that were not retrieved in the original search request and/or are not stored by the repository.

3.3.1 Textual Search (KMI, Mímir)

The Textual Search Wrapper uses the Khresmoi Mímir Interface (KMI) to post queries to a Mímir index [14]. It connects to an instance of the Textual Search SCA in the integration layer[4], the URL of which can be configured in the wrapper's config file. The internal EZDL queries are serialized and then send to the KMI, which converts them into a Mímir query. If the number of results returned is lower than a configurable threshold and the original query is not in English, the query is translated and send again. The results are merged, with documents in the original query language being boosted.

When filling document details, the textual content of the result document is retrieved through the integration layer[5] and then summarized using the automatic summarizer described in deliverable D4.4 [17].

3.3.2 Article Search (MedSearch)

The Medsearch Wrapper can be used to search for Pubmed articles[6]. EZDL queries are converted to Lucene syntax. The original search result contains only the URLs of the first two images contained in an article, which can be used to show a thumbnail in the results. On a request to fill details for a document, this wrapper fetches the full article details[7] potentially including additional images.

3.3.3 Image Search (ParaDISE)

The ParaDISE Wrapper can be used to search for images using text, as well as positive or negative image examples. EZDL queries are converted to the syntax supported by the Image Search SCA in the integration layer[8]. The wrapper provides no additional details on fill requests, since the original search result already contains all necessary information.

[4] http://atos1-khresmoi.ms.mff.cuni.cz:8080/khresmoi-textual-search/rest/documents
[5] http://atos2-khresmoi.ms.mff.cuni.cz:8080/khresmoi-mimir/rest/mimir/khresmoi/doc/search/renderDocument
[6] http://fast.hevs.ch:8080/MedSearch/res/textual/search/searchText
[7] http://fast.hevs.ch:8080/MedSearch/res/textual/articleDetails/
[8] http://atos1-khresmoi.ms.mff.cuni.cz:8080/khresmoi-image-search/rest/images/2D/searchImages

3.4 Objects and Fields

Internally, result objects are represented by subclasses of DLObject. These digital library objects can be used to describe all types of objects that occur in the context of search and storage, such as documents, images, persons, ontology terms, or archived queries. All DL objects have a unique, system-wide and immutable identifier (the OID), which is set by the creating class, as soon as a new DL object is initialized (e.g., by a wrapper or by the personal library). A DL object can have additional identifiers, such as a source ID (the identifier used by the original source, e.g. ParaDISE or Mímir), a URI or URL, a concept identifier, an ISBN or DOI. DL objects that share an identifier can be merged (when merging all original identifiers are kept).

In addition to an OID, and a set of identifiers, all subclasses allow a specific set of Fields. These fields describe objects of that subclass. Fields are type-safe and each field specifies the allowed (serializable) Java class for its values. All DL objects can be annotated with user comments that are handled by the Annotation Agent described above. Terms can additionally be annotated with semantic concepts.

Document: TITLE, AUTHOR, PUBLICATION_YEAR, ABSTRACT, ORIGINAL_SOURCE

TextDocument: all fields from Document, plus BOOKTITLE, BINDING, DOI, ISBN, ISSN, JOURNAL, LANGUAGE, NUMBER, PAGES, VOLUME, PUBLISHER, MEDIA_THUMBNAIL_URIS (URIs for thumbnails of images and other media represented in the document), PUBLICATION_DATE (a more precise representation of the publication date, including month and possible day), FREE_ACCESS (information about access restrictions), CLASS_LABELS (classification labels assigned to the document), TYPE (type of the document, e.g. article, book, webpage)

MimirDocument: all fields from Document, plus MIMIR_SNIPPET (representative snippet from document), MIMIR_SNIPPETLIST (list of snippets with hits), MIMIR_CATEGORIES (category of result document), MIMIR_ORGANIZATION (publishing organization or web site), MIMIR_TRUSTWORTHINESS (numerical value of estimated trustworthiness), MIMIR_AUDIENCE (target audience of document), MIMIR_ID (internal ID), MIMIR_IDTYPE (internal ID type, e.g. URL or PubMed ID), MIMIR_IMAGE_URIS (URIs of all images found on a result document), PUBLISHER, CLASS_LABELS, DOI, LANGUAGE, PUBLICATION_DATE, MIMIR_READABILITY (numerical value of estimated readability), MIMIR_ATTRIBUTION (numerical value of attribution), ADDRESS_COUNTRY (publication location of result document), ADDRESS_CITY (publication location of result document), SUMMARY (generated summary)

ImageDocument: all fields from Document, plus IMAGE (the binary image), URIS (a list of URIs), IMAGE_SIZE (image size), IMAGE_COLOR_MODEL (color model), IMAGE_BINARY_FORMAT (the format of the image, e.g. JPEG or GIF), IMAGE_THUMBNAIL (a binary thumbnail of the image), IMAGE_THUMBNAIL_URI (the URI of the thumbnail image), IMAGE_DESCRIPTION (a textual description of the image, e.g. the image caption), LANGUAGE (language used in image description), CLASS_LABELS (classification labels assigned to the image), ONTO_MODALITY (modality of the image), ONTO_ANATOMY (anatomical concepts represented in the image), ONTO_PATHOLOGY (pathology concepts represented in the image), PUBLISHER (publisher of the image)

OntologyTerm: LABEL (the preferred label of the term), DEFINITION (a definition of the term), ALTERNATIVE_NAMES (alternative labels), CLASS_LABELS (classification labels assigned to the term),

Person: FIRST_NAME, LAST_NAME, NAME_POSTFIX, EMAIL, NICK_NAME, PREFERRED_LANGUAGE, KNOWLEDGE_LEVEL, BIRTH_DATE, GENDER, DESCRIPTION, COUNTRY_OF_ORIGIN, PRIVACY, PRIVILEGES, PROFILE_IMAGE, MEDIA_THUMBNAIL_URIS

HistoricQuery: Historic queries from users' search histories are handled slightly different than other DL objects, since they don't have fields, but instead act as a wrapper class for a query tree, the timestamp of the original query and the result count.

4 Frontends

Both the desktop and the browser frontend use the concept of perspectives to provide task-based selections and configurations of tools. Perspectives can be preconfigured by the system administrator (and the desktop client

comes with perspectives for searching, for image searching or browsing, and for organizing and sharing documents with other users), or can be configured and persisted by the user. Each tool represents a specific set of (logically connected) functions and is independent of all other tools.

Tools communicate by the way of an internal dispatcher. They send event messages through the dispatcher and other tools that provide the requested functionality can register for specific events: for instance, clicking on a result in the Search Tool triggers a DetailEvent and all tools that have registered for this event can react to this trigger.

4.1 Available tools

The following section describes the various tools of the Khresmoi Professional desktop, mobile and browser client. Some tools are only provided for the more comprehensive desktop version.

4.1.1 Search Tool (desktop, browser, mobile)

The Search Tool provides the main functionality for the Khresmoi Professional clients. It implements a variety of different query forms for specific purposes: a simple search bar, an extensible search form with advanced fields (year range, authors, language, etc.), and an image search form with a text field, a modality chooser and a canvas that allows specification of positive and negative examples. All queries are transformed into an internal query tree (as described in deliverable D3.3 [9]). A source selection view allows selection of the libraries or information sources to be searched (see Fig. 4).

Different result views can be added to the search tool to show search results in list or grid form (see Fig. 6). The result views offer multiple functions to manipulate results: sorting and grouping according to different criteria, filtering by configurable result facets or free text (including inverse filters), or extracting frequent terms, authors or other features to be visualized as a list, bar chart or term cloud. Facets or extraction strategies can be easily replaced or added.

4.1.2 Tray (desktop, browser, mobile)

The Tray allows temporary storage of relevant documents for the current search session as part of a berrypicking strategy [3]. The content of the tray can be filtered, exported into different bibliographic formats (such as BibTeX or RIS), and for registered users persisted in the personal library (see Fig. 5).

4.1.3 Personal Library (desktop, browser)

The Personal Library was described in deliverable D3.4 [13] and provides a personal, taggable object repository. Documents, images, authors, search terms, queries and other objects can be stored in the personal library. Users can add (hierarchical) tags to organize their collection. Stored objects can be shared with other users (see Fig. 7).

4.1.4 Detail Views (desktop, browser, mobile)

The Detail Tool shows document details for results. The exact details shown can be configured by the user, with the following options being available: title, author and publication year, alternate labels (for concepts), classifications, abstract, snippets, generated summary, source links, fulltext URL, thumbnails, and a term cloud of common terms (see Fig. 8).

The detail view offers a large number of operations that can be performed on displayed documents either through button triggers or links within the detail document.

- printing the document
- copying the document to the Tray
- storing the document in the Personal Library
- opening the document in a new internal tab
- exporting the document into different bibliographic formats (e.g., BibTeX or RIS)
- switching to the previous/next document in the display history
- opening the full text for the document in a browser

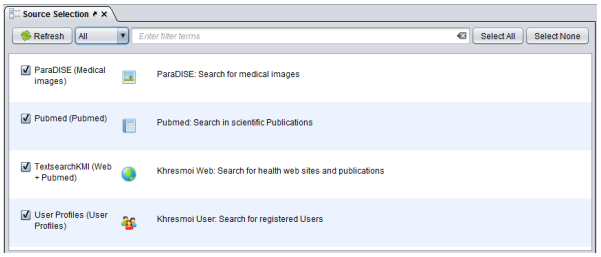

Figure 4: The source selection view.

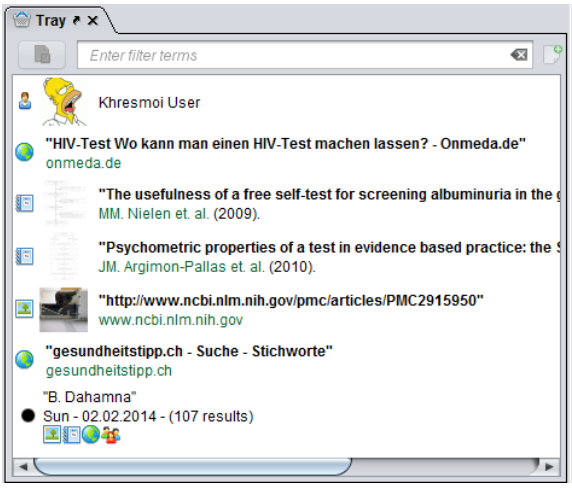

Figure 5: The tray tool.

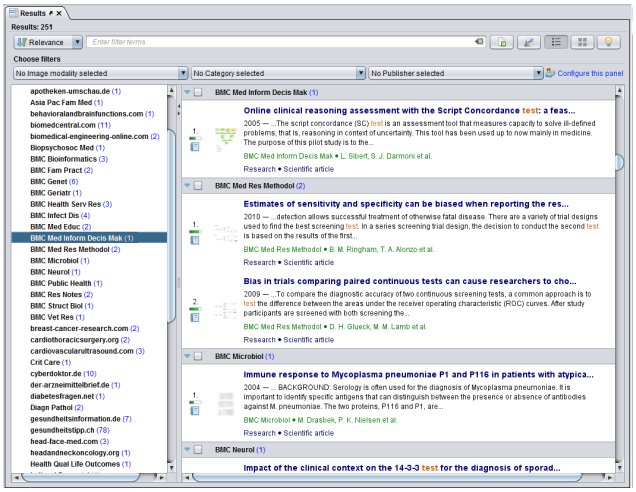

Figure 6: The result view.

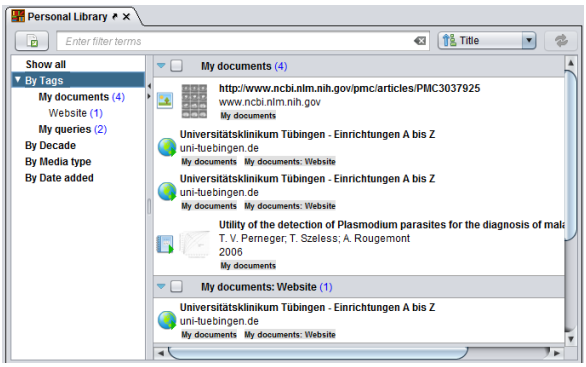

Figure 7: The personal library with hierarchical tags.

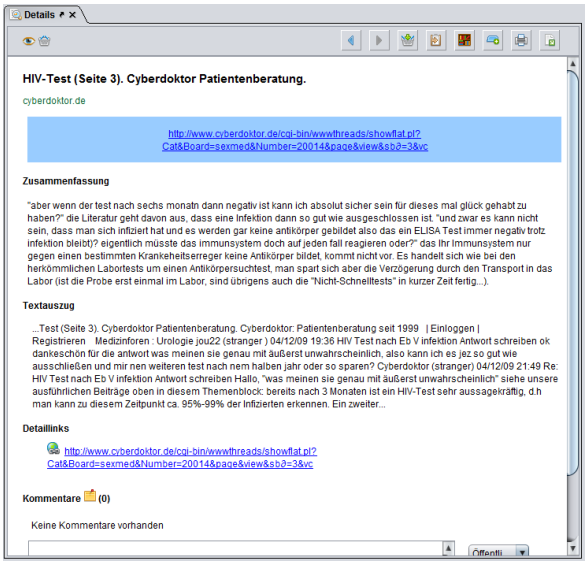

Figure 8: The detail view.

- opening the profile of a commenting user
- searching for an author or term from the document
- translating a foreign-language document into the current user's preferred language
- translating smaller sections of a foreign-language document and update the translations
- annotating the document with public or private comments
- searching for similar images
- adding an image from the document as to the query

In addition, for image documents basic image manipulation functions are provided (panning, zooming, and changing of contrast and brightness).

4.1.5 Group Management Tool (desktop)

The Group Management Tool allows creation, update and management of user groups (see Fig. 9). Users can either join an existing group or create a new group that they administrate. Groups can be public (and anyone can join) or private for a predefined set of users. Currently these groups can be used to share documents or annotations.

4.1.6 Suggestion Tool (desktop)

The Suggestion Tool presents search continuation suggestions requested for the current search situation (see Fig. 12). It is described in detail in [25, 26]. The search suggestions are adapted for the user's current situation and most can be executed by double click. Explicit positive and negative feedback can be given by the user.

4.1.7 Query History Tool (desktop, browser, mobile)

The Query History Tool provides a list of past search queries (see Fig. 12). For guest users this list only covers the current session, but registered users are presented with their past 200 queries and they can additionally fetch older queries. The old queries are presented grouped by date, can be filtered using free-text, re-used, combined and stored in the personal collection of registered users.

4.1.8 Extraction Tool (desktop, browser, mobile)

The Extraction Tool can be triggered from a result list and presents frequent terms, authors or other aspects of the result either as a list, as a bar chart or a term cloud (see Fig. 10). Clicking on an extracted term or author filters the current result list. Double clicking starts a new search for this author or term. Additionally, the extracted aspects can be dragged to the tray, personal library or a query form (where they will be combined with the current query).

4.2 Personalization

Users can personalize their search experience using the Config framework. Within the desktop or browser client, user configuration is possible through the preferences or options dialog, a tabbed dialog where each tab represents a group of configuration options (see Fig. 14). The same dialog also allows editing of the personal user profile for registered users (see Fig. 13), where information such as name, gender, birthdate, preferred language, home country, experience level, a profile picture or a profile description can be set (Fig. 15 shows the profile as seen by another user).

The ConfigFactory provides different strategies for each of the currently three supported configuration *contexts*.

- The user-wide configuration context is used for all configuration options that are specific to one registered user, and should be kept the same, no matter which client or machine is used to connect to Khresmoi Professional. Profile information is an obvious example for the user context.

- The machine-wide configuration context allows for configuration that is specific to a particular machine and client, and is shared by all users of that machine. Screen configuration or connection speed are examples for configuration options in the machine context.

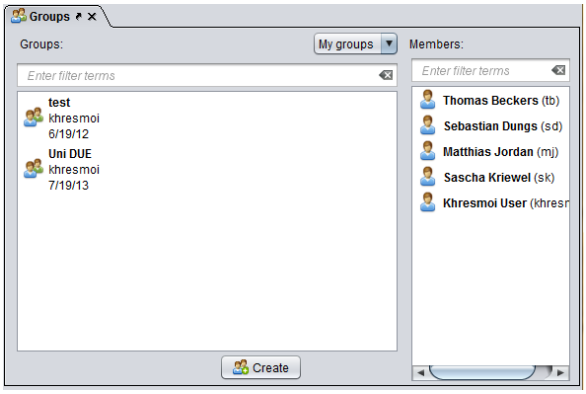

Figure 9: Management of user groups.

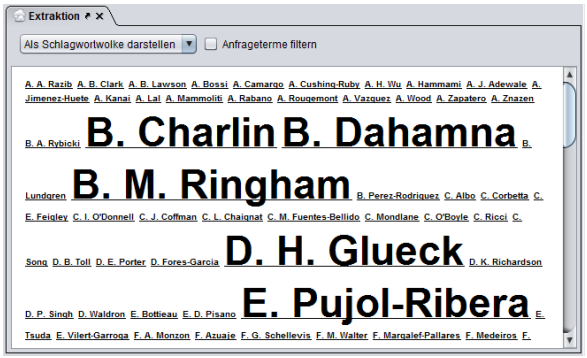

Figure 10: Extraction view as term cloud.

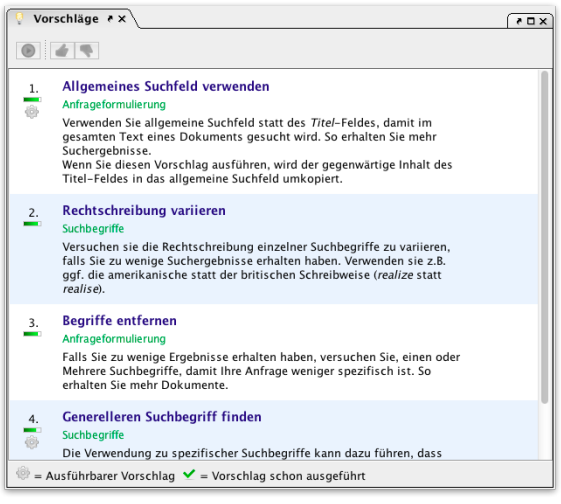

Figure 11: Search suggestions.

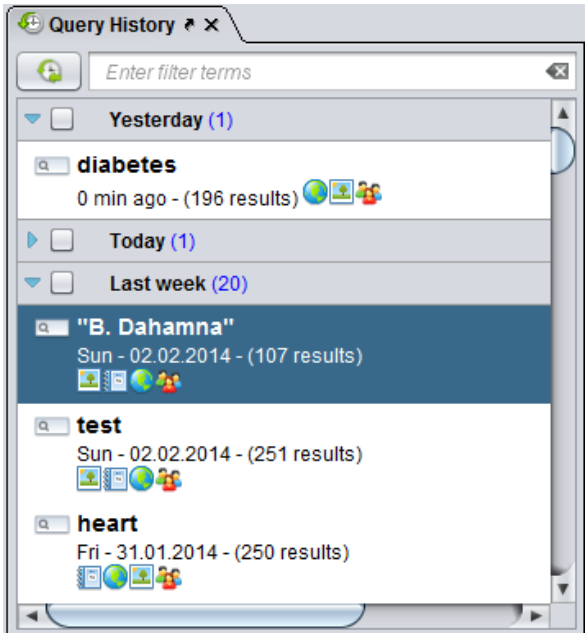

Figure 12: Query history.

- The role-wide configuration can be used to differentiate between individual roles that a user can assume. For instance, a user might have both the role “physician” and “admin” using the same account and can set configuration options specific to one of their roles.

All configuration options are identified by their context, a namespace (which is used to group options by tool or function) and a localname. Two strategies for storage are currently supported. The local store strategy uses a local directory to store options in Java `.properties` files. All keys for a specific namespace are collected in one file (using the namespace as the filename). The agent store strategy uses the User Agent to store configuration options on the server (using a relational database).

4.3 Clients

Three different clients have been provided for Khresmoi Professional, all using the framework described herein. A specialized client for radiologists (based on the desktop frontend) has also been developed.

4.3.1 Desktop version

The desktop version of the Khresmoi Professional frontend has been implemented in Java Swing and is delivered through Java Webstart (see Fig 16). The client offers a search desktop and has the most comprehensive set of tools and support functionalities of the clients. It connects to the backend framework through an MTA.

In addition to the tools, which are arranged into a number of pre-defined and/or customizable perspectives as described above, the desktop client also offers an application menu. Through the application menu users can open new tools, access different views connected to tools (such as individual result lists), switch between perspectives, save and load customized perspectives, import documents from BibTeX, configure the client, access the help system or close the application.

The tool area on the bottom left shows the tools which are active in the current perspective and the icons allow for quickly switching between tools, opening new tools or using them as drop targets for documents (so that a document could be dropped into the personal library or the tray without the tool being open and taking up screen estate). A small info area on the bottom right shows the current location if detected, the interface language and the preferred language of the user. In between those two area is the status bar which gives textual status updates on system actions.

In addition to drag and drop interaction, nearly all objects also provide context menus. For example, the context menu on result items allows them to be stored in the personal library, send to the tray, added as positive example to a query or opened in a new internal tab.

4.3.2 Radiology version

The Khresmoi Radiology frontend (see Fig. 19) is a customized version of the Khresmoi Professional desktop frontend. It has been adapted to be deployed in a hospital radiology department [21]. Special faintly illuminated workspaces are used by radiologists while diagnosing patient conditions using three-dimensional medical images. Therefore, a different colour schema was used for the interface, mainly utilizing different tones of grey to prevent light pollution. Since the interface is tailored to a special use case it features unique tools and perspectives. The *Index Tool* allows the user to browse through the hospital case base. When a case is selected from the index, the actual image data is retrieved and shown in the interface. The data is loaded on demand due to the high file size. The system features several ways to interact with image data. Users can scroll through an image series, zoom into an image or relocate it on screen. Additionally, users can mark areas in the image data to indicate interest in a particular region in any slice of an image series. The system then retrieves image data of similar cases and additional textual resources.

The Radiology frontend also includes most features of the Khresmoi Professional desktop frontend, such as the personal library and the collaborative tools. Therefore, users can easily transition between systems.

4.3.3 Browser version

The frontend for web browsers has been implemented as a web app (RIA, rich internet application) using Google Web Toolkit (GWT). It has been described in [5, 10].

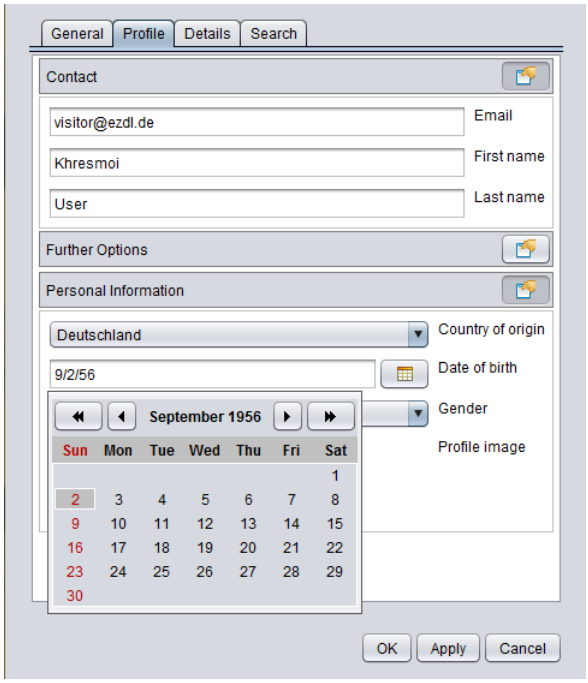

Figure 13: Adding personal details to profile.

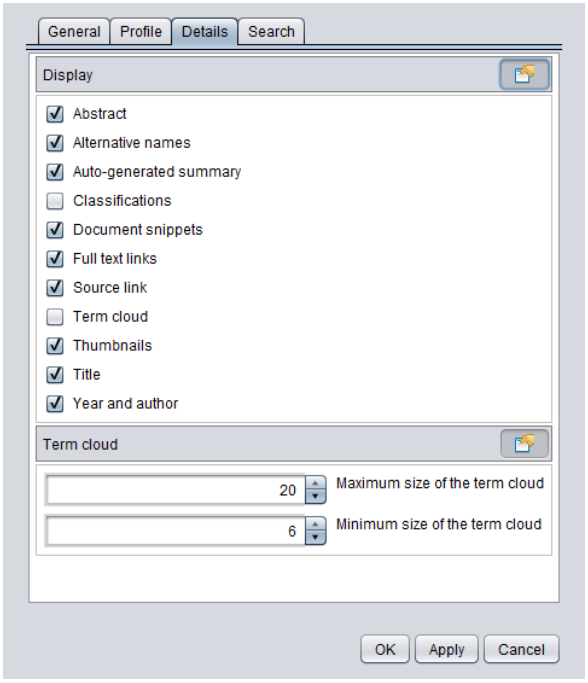

Figure 14: Configuring details for documents.

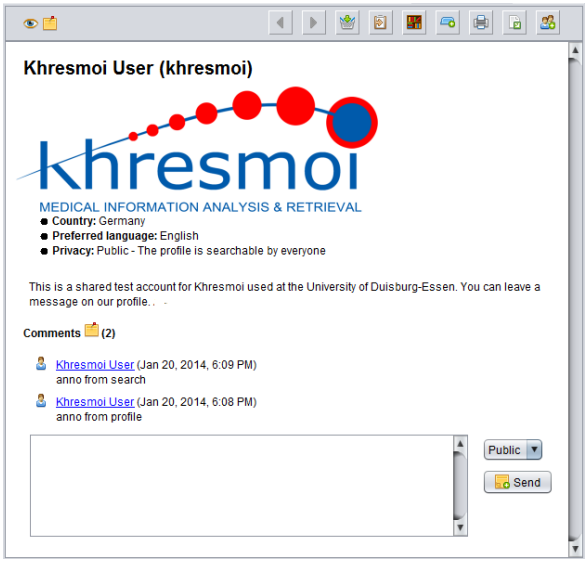

Figure 15: User profile with comments.

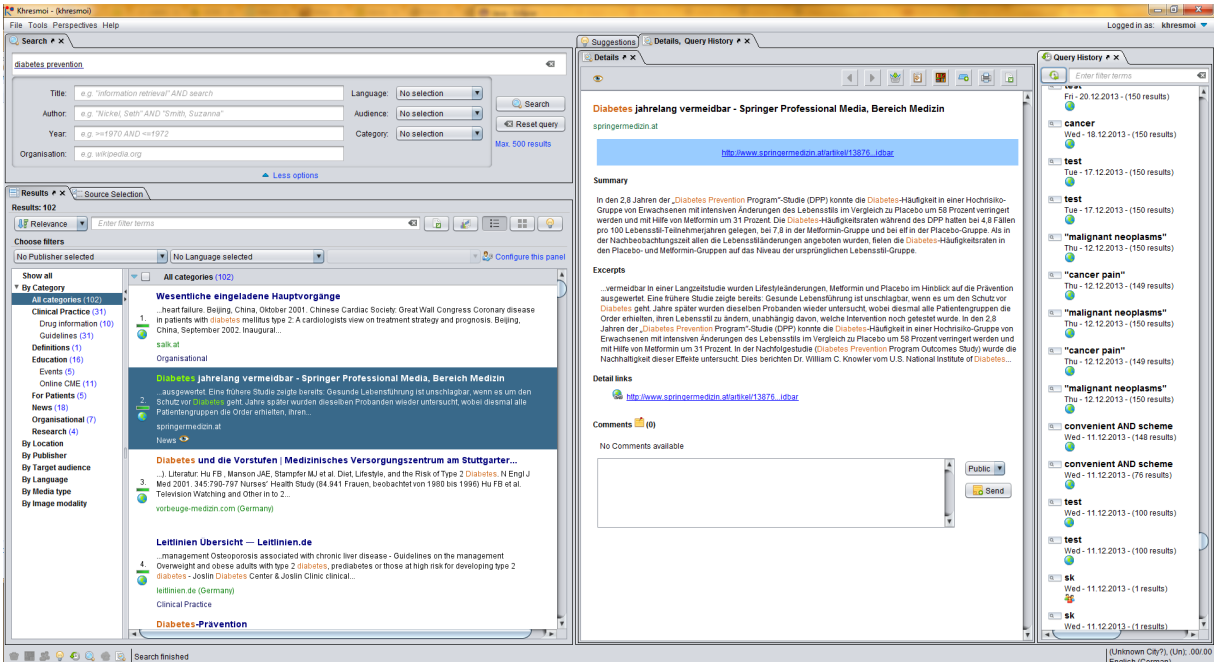

Figure 16: The desktop version of the client.

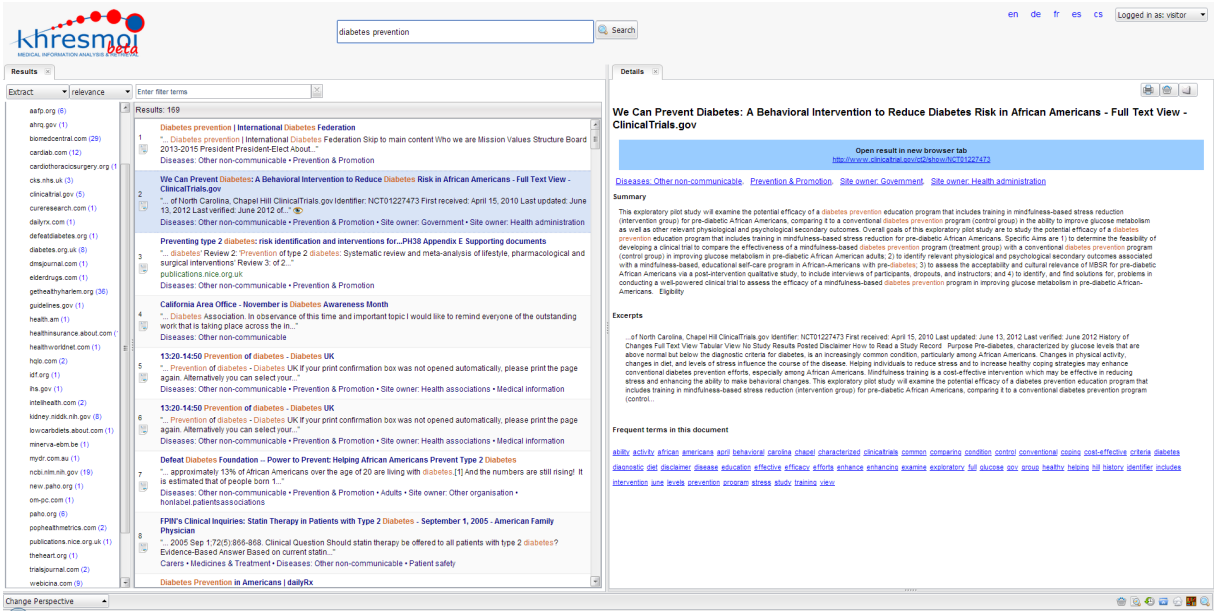

Figure 17: The browser version of the client.

As an RIA, the web frontend uses common web technologies to achieve a look and feel that is closer to a traditional desktop application than to a static web page (see Fig 17). It allows direct manipulation by drag and drop and offers context menu for many objects. Most of the tools available for the more comprehensive desktop client have been ported to the browser application, but some functions have been removed to make it simpler.

4.3.4 Mobile version

The mobile frontend for Khresmoi Professional is described in [8, 22]. It has been implemented as an app for Android 4.2 and higher, and is available from the Google Play Store.

Due to the smaller screen estate available the concept of perspectives does not translate to mobile devices. As for the desktop and browser version, connected functionalities are grouped into tools, but only one tool (for smartphones, see Fig 18) or two related tools (for larger tablet devices) are shown at a time. Navigation between tools is provided through the *action bar* at the top of the screen. Switching between apps will not clear the tray, instead the contents are stored on the phone (and can be pushed to the personal library for registered users). Documents can also be exported and shared using the default sharing intent of the Android OS.

A REST-based web service handles the communication between the Android app and the EZDL backend. This additional communication layer was introduced to deal with the inherent instability of mobile connections. The web service acts as a proxy that translates REST requests (using the JSON format) into the messages used by the backend and translates the responses back into JSON objects for the app.

5 Installation and configuration

The Khresmoi Professional backend can be installed on any Unix-related operating system. The individual agents of the backend can be run on seperate machines or use the same machine,

The prerequisites for installation are Java 7, Maven, and MySQL server 5. For easy delivery of the webstart client some kind of web server (such as the Apache HTTP server[9]) is recommended. For installation of the web client software a servlet container such as Apache Tomcat will be necessary [10].

As the first step of installation, a database needs to be created on each machine that runs an agent with a database backed store (currently these are the User, Repository, Personal Library, Suggestion and User Log agent). Scripts to create the necessary tables are bundled with the agent code. Database host, port, name, user and password need to be set in the main agent configuration file (see below).

Start scripts for all agents have been bundled with the software. They allow either the full, staggered start of the complete system, or starting individual agents if they are to be run on different machines.

The start configuration specifies the agent to start, the file prefix for agent configuration files and log files, and the name that is used to register the agent with the Directory Agent (see Listing 1). Multiple instances of a particular agent can be registered with the directory using different agent names. A detailed installation and start guide can be found in [15].

Listing 1: Example ant configuration for starting the Khresmoi MTA.

```
<target name="startMTA">
    <antcall target="startAgent">
        <param name="configDir" value="${configDir}" />
        <param name="filePrefix" value="mta" />
        <param name="subDir" value="service-mta-khresmoi" />
        <param name="agentClass"
               value="de.unidue.inf.is.ezdl.dlservices.backbone.mta.MTA" />
        <param name="agentName" value="MTA" />
     </antcall>
</target>
```

Agents can be further configured using plain text configuration files. These files should be placed in a directory named `.ezdl-khresmoi` located in the home directory of the user which is used to start the backend. A base agent configuration (see Listing 2) specifies the database access parameters, connection method and parameters

[9] https://httpd.apache.org/
[10] http://tomcat.apache.org/

and logging parameters for all agents running on that machine (but could potentially be overwritten by specific agent configurations).

Listing 2: Example base agent configuration (agent.properties).

```
# agent connector type
connector=jms
# JMS agent connector
jms.provider.url=tcp://localhost:61616
# max size for the agent log (available via directory web admin site)
log.maxsize=100
# name of the directory
agentname.directory=Directory
% database connection properties
db.url=jdbc:mysql://host:port/database
db.user=database user name
db.password=database user's password
```

Many agents, in particular wrapper agents, need or allow further configuration. For wrappers (see Listing 3 for an example) labels and categories to be used for display in the clients can be defined. This allows wrappers to be added to the backend without any changes to the clients. The wrapper configuration below also allows configuration of URLs for searching and fetching full texts, thus making it possible to easily switch between different search indexes or to use multiple wrappers of this type backed by different indexes.

Listing 3: Example wrapper configuration (wrappername.properties).

```
# class that implements the wrapper
wrapperclass=de.unidue.....KMITextualSearchWrapper
# number of concurrent sessions allowed
maxSessions=10
# how many results to request per page
fetchMaxResults=100
# how many result pages to request
fetchMaxPages=1

info.remotename=Websearch
% category label for source selection
info.category=med
% internationalized display name for category
info.category.de=Medizin
info.category.en=Medicine
% internationalized display name for source
info.description.de=Websuche
info.description.en=Websearch
% time
info.minimumTimeoutS=5
# URLs
url.search=http://atos1-khresmoi.ms.mff.cuni.cz:8080/...
url.fulltext=http://atos2-khresmoi.ms.mff.cuni.cz:8080/...
```

5.1 Plugin architecture

A plugin architecture allows extending the agents as well as the desktop client. Plugins are independent Java components which can be registered with a central component that later returns available plugins on request.

There are two ways of registering plugins: one is using the OSGi subsystem, that reads plugins from OSGi bundles residing in a dedicated directory. This is the primary way for users to extend the framework without changing the code base. The other way is to register the plugin from within the program ("static registering"). While any code segment of the framework can in principle statically register a plugin, the usual way is to have the start up code register code modules that extend the core functionality. Examples for this are the modules that

provide proactive suggestions. Also, the component that keeps information on the available digital libraries, the library choice view, registers a plugin statically that provides a mechanism to retrieve icons for each digital library by using a special URL.

Regardless of how the plugins are registered, the registration submits a plugin descriptor object that contains information on the kind of the plugin (its class), a textual description for logging, and an object that implements the plugin's business logic. A special type of plugins ("Runnable") is not registered in the central component but executed immediately upon registration. This provides a way to start any desired code at the start of ezDL. One use case is inserting test code that sleeps a short time and triggers a certain failure after waking up.

Clients can ask the plugin system for plugins that implement a given class and get a list of objects of this class in return. For example, the code that initializes the extraction function in the result list asks for extraction plugins and uses the objects returned for initializing the drop down list of available extractors.

6 Conclusions

This deliverable presented a flexible user interface framework for Khresmoi, based on the EZDL system. The EZDL framework has been extended in many ways. Two entirely new front ends (browser based and mobile) are provided. The previously DL centric EZDL software has been extended to deal with new document types such as images, ontology concepts, or web pages. User configuration of many aspects of the framework is now possible, and a plugin infrastructure provides further possibilities for easy extension with new features.

A number of additional agents have been implemented to deal with personal storage for collecting documents and queries, group management, sharing of documents, and user search, proactive query suggestions, search continuation support, creating and managing annotations retrieving and updating translations, and semantic lookup. Wrappers to search for images including relevance feedback and web pages have also been added to the original EZDL software.

In addition to the extension of the framework with new features and components, the existing framework has been made extremely modular, so that individual tools, search forms, strategies (e.g. for rendering results, export, query suggestions, facet extraction) can be easily replaced and recombined. The current framework is entirely open source [11] and available as individual Maven projects[12].

[11] The source code is licensed under GNU General Public License, version 3, and can be downloaded at `http://hg-pub.is.inf.uni-due.de/hg/data/ezdl/` and `https://hg-khresmoi.is.inf.uni-due.de/hg/ezdl-khresmoi`

[12] `http://mvn.is.inf.uni-due.de:8081/nexus/content/repositories/snapshots/de/unidue/inf/is/ezdl/`

7 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. The design of browsing and berrypicking techniques for the online search interface. *Online Review*, 13(5):407–424, 1989. `http://www.gseis.ucla.edu/faculty/bates/berrypicking.html`.

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

[5] Thomas Beckers, Sebastian Dungs, Norbert Fuhr, Lorraine Goeuriot, Jessica Ignalski, Matthias Jordan, Liadh Kelly, and Sascha Kriewel. D3.2 - Report on results of the WP3 first evaluation phase, August 2012. Public deliverable for the Khresmoi project.

[6] 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.

[7] Thomas Beckers, Sebastian Dungs, Norbert Fuhr, Matthias Jordan, and Sascha Kriewel. ezDL: An interactive IR framework, search tool, and evaluation system, 2014. Submitted for publication.

[8] Celia Boyer, Natalia Pletneva, Nolan Lawson, Sascha Kriewel, Thomas Beckers, Marlene Kritz, and Manfred Gschwandtner. D8.4 - Report on and prototype of mobile information access system. Technical report, August 2013. Public deliverable for the Khresmoi project.

[9] Sebastian Dungs, Thomas Beckers, Matthias Jordan, Sascha Kriewel, Nolan Lawson, Ivan Martinez Rodriguez, Andreas Tacke, and Miguel Angel Tinte Garcia. D3.3 - Report on query specification, result presentation and personalization, February 2013. Public deliverable for the Khresmoi project.

[10] Markus Franitza, Sascha Kriewel, and Sebastian Dungs. Entwicklung und evaluierung einer rich internet application für die suche nach medizinischen informationen. In *Proceedings of the IR Workshop at LWA 2012, Dortmund, Germany*, September 2012.

[11] 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.

[12] Lorraine Goeuriot, Gareth Jones, Liadh Kelly, Sascha Kriewel, and Pavel Pecina. D3.1 - Report on and prototype of the translation support, May 2012. Public deliverable for the Khresmoi project.

[13] Lorraine Goeuriot, Liadh Kelly, Sascha Kriewel, Gareth Jones, Ivan Martinez Rodriguez, Miguel Angel Tinte Garcia, Pavel Pecina, Ales Tamchyna, and Ren Donner. D3.4 - Report on user input for updating resources, August 2013. Public deliverable for the Khresmoi project.

[14] Mark A Greenwood, Valentin Tablan, and Diana Maynard. GATE Mímir: Answering questions google cant. In *Proceedings of the 10th International Semantic Web Conference (ISWC2011)*, 2011.

[15] Matthias Jordan and Thomas Beckers. ezDL - Developer Guide, 2014. `http://gimlet.is.inf.uni-due.de/ezdl/developer-guide.pdf`.

[16] Matthias Jordan and Thomas Beckers. ezDL - Operator Guide, 2014. `http://gimlet.is.inf.uni-due.de/ezdl/1.9.x/operator-guide.pdf`.

[17] Liadh Kelly, Johannes Leveling, Shane McQuillan, Sascha Kriewel, Lorraine Goeuriot, and Gareth Jones. D4.4 - Report on summarization techniques, February 2013. Public deliverable for the Khresmoi project.

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

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

[20] 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. available at `http://www.dlib.org/dlib/june04/kriewel/06kriewel.html`.

[21] Dimitrios Markonis, René Donner, Markus Holzer, Thomas Schlegl, Sebastian Dungs, Sascha Kriewel, Georg Langs, and Henning Müller. A visual information retrieval system for radiology reports and the medical literature. In *The 20th Anniversary International Conference on MultiMedia Modeling (MMM 2014), Dublin, Ireland*, January 2014. demo, accepted for publication.

[22] Stefan Muno, Thomas Beckers, and Sascha Kriewel. Konzeption und Implementierung einer Android-App für das ezDL-System. In *Proceedings of the IR Workshop at Lernen, Wissen, Adaption (LWA 2013)*, October 2013.

[23] Andreas Paepcke. Digital libraries: Searching is not enough–what we learned on-site. *D-Lib Magazine*, 2(5), May 1996. `http://www.dlib.org/dlib/may96/stanford/05paepcke.html`.

[24] 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 Andreas Rauber, Stavros Christodoulakis, and A Min 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.

[25] Andreas Tacke and Sascha Kriewel. Strategische Suchunterstützung auf Makro- und Mikroebene. In *Proc IR Workshop at Lernen, Wissen, Adaption (LWA 2013)*, Oktober 2013.

[26] Andreas Tacke and Sascha Kriewel. Strategic search support on macro and micro level. *Datenbank-Spektrum*, 2014. Manuscript accepted for publication.

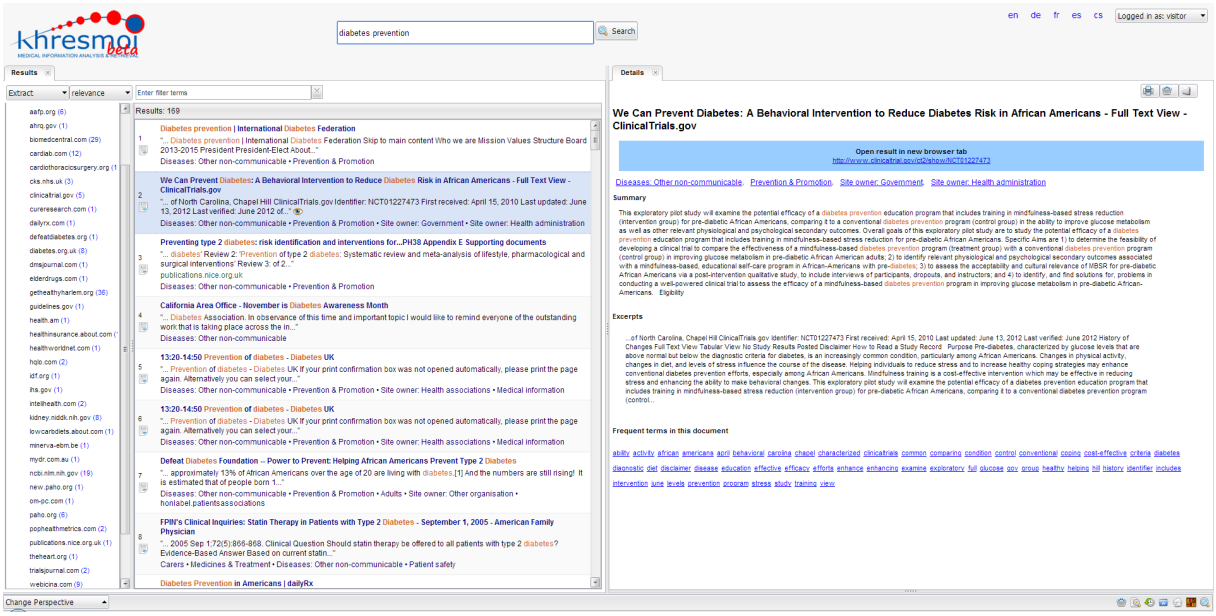

Figure 18: The mobile version of the client.

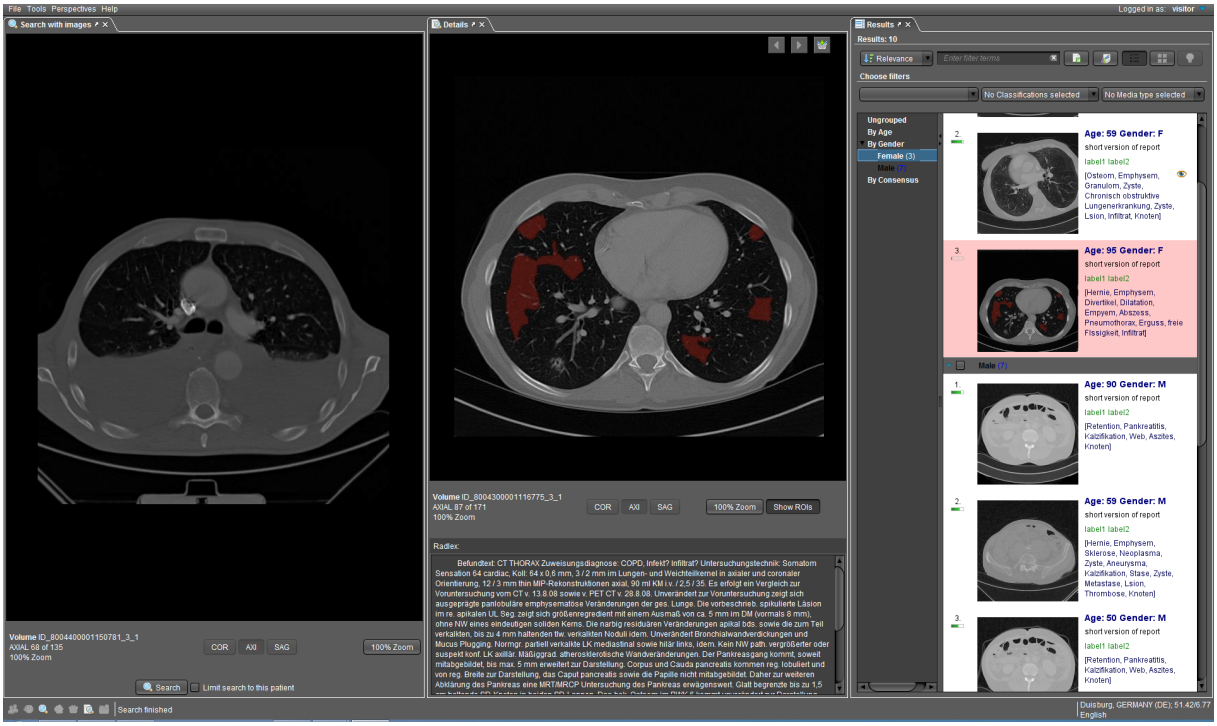

Figure 19: The radiology version of the client.