

Medex: (Medical Expert)

Powered by Context Targeted Search Engine

Lope Beltran II

Email:beltranlope@gmail.com

Jaybee Sarmiento

Email:jaybeesarmiento@gmail.com

Danille Torrecampo

Email:danielle.torrecampo@gmail.com

Adviser: Ramon Achilles De Guzman

Networking and Distributed Systems Group

Department of Computer Science

College of Engineering

University of the Philippines Diliman

Email:kyldeguzman@gmail.com

March 2010

Abstract

This paper shows the implementation of a dynamic system that can search existing records in a database relevant to the given record. This prototype is applied to the medical field, specifically a medical record database. The system is XML-based and uses XQuery as its underlying search engine core. To fully utilize XQuery, we use eXist, a Java-based, open source, native XML database. Existing methods such as data mining, Keyword in Context (KWIC), etc. are powerful tools and are available to most of the people.

The paper proposes to use these tools and create a dynamic system that delivers related results according to the given data so that it can be used in different fields. The design of this program allows it to be configured for other uses, like searching relevant police records. For now, it would focus on the health record domain.

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 3 |
| 2 | Background Information | 5 |
| 3 | Literature Review | 6 |
| 3.1 | Data/Content Classification and Filtering | 6 |
| 3.1.1 | SOAP Format | 6 |
| 3.1.2 | KWiC | 7 |
| 3.1.3 | Synonymy and UMLS | 8 |
| 3.2 | Search Engine and Format Parser | 12 |
| 3.2.1 | XML Based Search Engine | 12 |
| 3.2.2 | eXist | 12 |
| 3.2.3 | Apache Lucene | 13 |
| 3.3 | Delivery | 14 |
| 3.3.1 | AJAX | 14 |
| 3.3.2 | The Yahoo! User Interface Library | 16 |
| 4 | Statement of the Problem | 16 |
| 5 | Methodology | 17 |
| 5.1 | Format Parser | 19 |
| 5.2 | Content/Data Classification and Filtering | 21 |
| 5.3 | Search Engine | 22 |

| | | |
|----------|--|-----------|
| 5.4 | Delivery | 23 |
| 6 | Results | 25 |
| 6.1 | Speed | 26 |
| 6.2 | Relevancy | 28 |
| 7 | Conclusion | 29 |
| 7.1 | Contributions of the Study | 30 |
| 7.2 | Future Research | 30 |
| A | Deployment | 33 |
| A.1 | Requirements for Deployment | 33 |
| A.2 | Installation for Deployment: (Windows) | 33 |
| A.3 | Search Engine Application Modification | 34 |
| A.4 | Tuning Exist | 38 |
| A.4.1 | Memory Settings | 38 |
| A.4.2 | Cache Settings | 38 |
| A.5 | Using the Application (MedEx Deployment) | 38 |
| B | Screenshots | 40 |

1 Introduction

In this age, technology has advanced further than what we expected. The appearance of databases has proved to make organizing data better and easier. Since the advent of computer systems, documents that were formerly filed and kept away in file cabinets have been encoded on databases, which allow access from anywhere thanks to the Internet.

Almost everyone is in touch with this acquired technology, including hospitals. Medical records of patients have been stored away in the world of bits and bytes (while keeping a hardcopy of it for backup). In case doctors and nurses need to look at the file of a previous patient, they could view it with a click of a button. Searching for specific cases has been less cumbersome. Converting medical record formats to another can simply be done, thus exchanging information can be less complicated.

Since search Engines today are mostly applied in searching websites given a keyword, why not change the scope to a certain domain to acquire relevant results in that domain? Why not deploy a system for that sole domain to act as a search engine? For example, given a medical record, why not let the system search for relevant records in the past for easy cross referencing of assessments and plans done instead of accessing records individually to see if it is actually relevant to the case at hand

The thesis aims to further improve the lives of medical practitioners with the use of context targeted search engine. Imagine a "Baryo" or "village"

scenario, which are very common in the Philippines, where access to modern day technology and medicine is very limited. Doctors in the baryo can use the proposed system to improve diagnoses and plan a more appropriate treatment for his or her patients. Here the dynamic system is used in a medical record scenario where given a medical record to the system, it searches its database or databases outside the domain to acquire relevant medical records to act as a second opinion for the doctor.

Given a sample record (i.e. symptoms, age, gender, observations, etc.) the system can search and output related medical cases for the medical practitioners to compare with. The proposed system can also output various statistical data related to the case (i.e. mortality rate, survival rate, most common diseases).

The proposed approach requires four parts.

1. System Format and Domain
2. Context Filtering/Classification of Data
3. Search Engine with Ranking of Results
4. Summary of the Results

These parts will further be discussed and elaborated in the methodology section.

In summary, the thesis makes at least two contributions

1. Creates a system for improving diagnoses and treatment of ill people.
2. Proposes a dynamic system for searching related results given input data and return appropriate and accurate results that can be applied to any record domain (e.g. medical, police).

2 Background Information

The thesis deals with creating a dynamic system for data classification with the ability to deliver appropriate results. With a few configurations, the system can be used in other fields.(i.e. delivering appropriate website advertisements, delivering relevant police records)

The desired system has four parts:

1. Format Parsing - it deals with the generation of all the configuration the system needs for the domain
2. Content/Data classification and Filtering - takes care of the acquisition of keywords to be used for searching.
3. Search Engine - the program searches and ranks appropriate results based from the keywords.
4. Delivery - the program outputs the results by rank, includes an easy view of a record and a summary of the results.

The proposed system uses an XML based search engine for the following

reasons. First, the XML format is dynamic and is a widely-used standard for data representation and exchange. It is also well structured, hierarchical, partly ordered and pervasively cross-linked. XQuery has a structured query[7] where it can practically have full control of the MVC framework, which is the main reason why the system is capable of being dynamic.

The system also uses the medical record format called S.O.A.P.(Story, Observation, Assessment, and Plan). This format is commonly used in various hospitals. S.O.A.P has a lot of variations but still sticks to its basic form.

3 Literature Review

This section outlines different technologies and methods used for the project. It also includes existing or similar systems.

3.1 Data/Content Classification and Filtering

3.1.1 SOAP Format

Hospitals, clinics and other medical facilities apply a standard for writing a patient's medical record. It is represented by the acronym SOAP[1], which stands for Story or Subjective, Observation or Objective, Assessment and Plan. The Story part contains the patient point-of-view of his health, what symptoms he felt, when did these start and how long has these been bothering him. The Objective part holds the person in charge's (a nurse

or a doctor) observation of the patient. This includes the weight and the bodily appearance of the patient. The diagnosis formulated from both the subjective and objective part is written under Assessment and the solutions to be provided are stated in Plan. Having this format allows easier search since querying can be done on a specific field.

For records to be more credible, a Final Pathological Diagnosis must be included. This ensures that there was no misdiagnosis or the misdiagnosis has been resolved.

3.1.2 KWiC

KWiC is an acronym for Key Word in Context and is most commonly used for concordance lines. A concordance is an alphabetical list of the principal words used in a book or body of work, with their immediate contexts. The algorithm was first coined by Hans Peter Luhn. The KWiC index system accepts an ordered set of lines, with each line being an ordered set of words, and each word being an ordered set of characters. This is also known as a permuted index.

It is used for the extraction of keywords within the context and attempts to find the segments that maximize the number of, in this case, medical terms[9].

3.1.3 Synonymy and UMLS

The Unified Medical Language System (UMLS)[2] is a metathesaurus which distributes key terminology, terms and resources to develop and improve medical related systems. This is created by the National Library of Medicine(NLM) to help develop and improve synonymy and understanding of terms in various computer systems. The UMLS converts terms into codes and serials so that machines can easily differentiate and categorize medical information. It contains vast biomedical information which can be categorized into multiple subtrees or subset. These subsets are submitted by known and highly acknowledged medical organizations. By default, UMLS knowledge sources are multi-purpose, meaning that it is not optimized for specific applications. It can be applied to a wide variety of systems with different functions concerning records, medical processes, and public health data. UMLS' subsets can be customized for the optimization of specific programs. There are three UMLS Knowledge Sources, namely: the Metathesaurus, Semantic Network, and Specialist Lexicon.

Metathesaurus :

The UMLS metathesaurus is a large database containing information about various health and medical concepts. The metathesaurus is built from various thesauri, code sets, terms about medicine which are referred to as "source vocabularies" organized into concepts and meanings. The source

vocabularies are recognized as U.S standards for health transactions in accordance to Health Insurance Portability and Accountability Act (HIPAA) or as target to U.S. government-wide clinical standards selected by the Consolidated Health Informatics eGov initiative.

Semantic Network :

The semantic network categorizes all terms and concepts within the Metathesaurus. It distinguishes relationships among these terms. The semantic network also provides semantic types which may be assigned to concepts which defines it relationship among other concepts. All concepts and terms in the metathesaurus is assigned into at least one semantic type. These types are defined as textual and heirarchical.

Specialist Lexicon :

The specialist lexicon is intended as a general English lexicon. The specialist lexicon analyzes terms syntactically, morphologically, and orthographically, which are needed by SPECIALIST Language Processing System. The Specialist Lexicon allows variability in the language. It can handle terms with different prefixes and suffixes. It also allows multi-word terms.

How It Works :

Each term in the UMLS are considered atoms (AUI). These atom can be part of different subsets of the UMLS. All the AUIs are linked to a single

string identifier (SUI) which represents the same occurrence of the string. All the strings are then linked to a term identifier (LUI) which contains the lexical variants of the string. The LUIs are linked to concept identifiers (CUI) which holds the meaning of the terms.

| AUI (Atoms) | SUI (Strings) | LUI (Terms) | CUI (Concepts) |
|-------------------------|---------------|-------------|----------------|
| Atrial Fibrillation | S0016668 | L0004238 | C0004238 |
| Atrial Fibrillations | S0016669 | (Synonyms) | |
| Auricular Fibrillation | S0016899 | L0004327 | |
| Auricular Fibrillations | S0016900 | (Synonyms) | |

Figure 1: The relationship of UMLS identifiers

All of these identifiers are important keys in the metathesaurus in mapping and customizing medical concepts. One example is using CUIs to search for all known terms for a specific concept.

How to Use UMLS :

In order to setup an environment and access the UMLS Knowledge Source Server, the following must be completed.

1. Obtaining a Simple Object Access Protocol (SOAP) 1.2 Compatible Engine (i.e Apache Axis 1.4).
2. Obtain the Authentication Web Services Description Language (WSDL) for the CAS Authentication web service and WSDL for the UMLSKS web service.

3. Generate Client stubs for the clients.
4. Develop and compile your application code and generated stubs.
5. Run the application.

Access to the UMLS such as searching CUIs are done through calls defined in the WSDL.

PROS of Using UMLS :

By using UMLS, applications can generalize biomedical terms and concepts. Systems can distinguish terminologies and define synonymous inputs given by the users generating more efficient and accurate results. The UMLS can also be customized and optimized for specific applications, i.e. using of certain subsets and sub-trees. UMLS also has multi-language support which can be very helpful in different countries.

CONS of Using UMLS :

Accessing and searching the large UMLS database might take some time and may slow down systems. Ambiguous terms can also be found in the metathesaurus, i.e. the term "cold" which might refer to the common cold, the temperature cold, or Chronic Obstructive Lung Disease, which might confuse the system.

3.2 Search Engine and Format Parser

3.2.1 XML Based Search Engine

XML, which stands for Extensible Markup Language, is a set of rules for encoding documents electronically. It is a textual data format and strongly emphasizes simplicity, generality and usability. Because of its ability to represent a mix of structure and unstructured data[8], it became widely used as a standard for data representation and exchange. It is also used for the representation of arbitrary data structures. XML is famous for its representation of data that can be used across various system.

3.2.2 eXist

eXist[5] is a Java-based, open-source native XML database with XQuery as its core. XQuery is typically used to query XML. eXist also contains Apache-Lucene, a framework for search engine applications. It also has a score of 99% on the XQuery Testing Score (XQTS), which ensures a higher hit rate of keywords. eXist also gives flexibility since a user can create his own index for collection of files which can be advantageous for searching. Using XQuery with an MVC framework, it gives full control of the model, which is an XML stored in eXist, the view, which is a xql file that contains XHTML, and the controller which is also in an XQuery language.

3.2.3 Apache Lucene

Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java [3]. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform. It also provides a module for indexing of XML data with analyzers that can identify the delimiters of the keywords. With this module, scoring is also made available for easy ranking of results.

Lucene has the following features:

- ranked searching – best results returned first
- many powerful query types: phrase queries, wildcard queries, proximity queries, range queries and more
- fielded searching (e.g., title, author, contents)
- date-range searching
- sorting by any field
- multiple-index searching with merged results
- allows simultaneous update and searching

3.3 Delivery

3.3.1 AJAX

AJAX is shorthand for Asynchronous Javascript and XML. It is not a new technology per se, but rather a combination of technologies that can be used together to build a more powerful web application[11]. By combining these technologies, a web application could imitate the richness and responsiveness of a desktop application.

AJAX uses the following technologies:

- XHTML and CSS for the interface.
- Document Object Model (DOM) for dynamic display and interaction.
- XML and XSLT for data manipulation.
- XMLHttpRequest for asynchronous data retrieval.
- Javascript as the glue that connects everything together.

In contrast to the classic web application model, which lets the user wait for a certain amount of time while the server does its processing, the AJAX model allows asynchronous user interaction with the application[10] (see figure 2). Using an AJAX engine as a mediator between the user and the server, it provides the user with the needed data from the server without refreshing the page. So whenever there is no need to contact the server, the AJAX engine provides the needed changes for the front end, thus saving

waiting time for the user. Most modern browsers support AJAX because

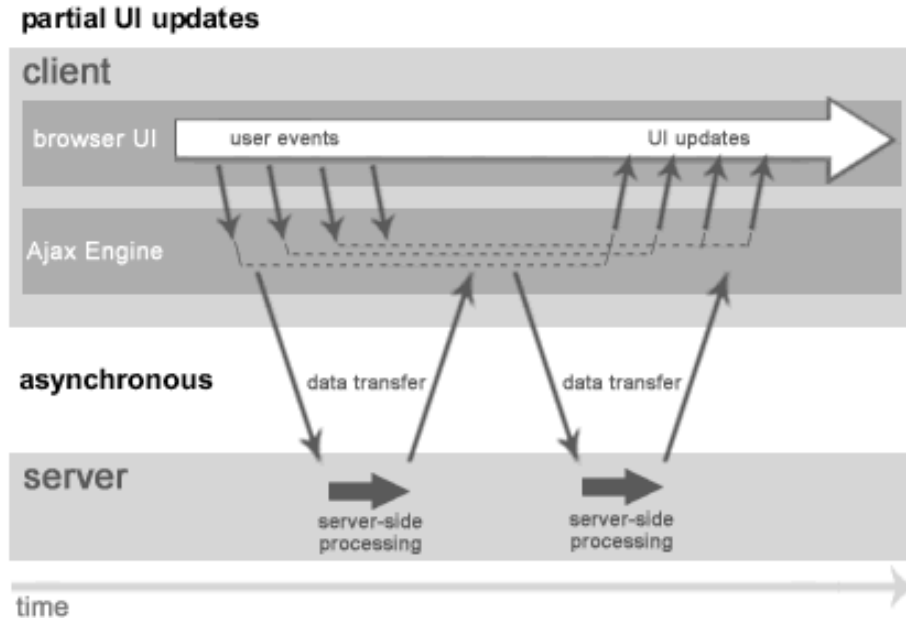


Figure 2: The asynchronous pattern of an AJAX application

the said technology does not need any other plugin or special software to run aside from those stated above [6]. What makes this approach easy to use is that most developers are already familiar with or perhaps skilled with the technologies making up AJAX. Because of its growing popularity, most companies have created AJAX frameworks. Examples of these frameworks are Google Web Toolkit (GWT) and Yahoo! User Interface (YUI), which is used for this project.

3.3.2 The Yahoo! User Interface Library

Being one of the major supporters of AJAX, Yahoo! has created their own AJAX framework. The YUI Library[4], a collaboration between front-end developers around the world, is a set of utilities and controls written with JavaScript and CSS for interactive web applications. It uses techniques such as DOM scripting, DHTML and AJAX. This AJAX framework is proven, scalable, fast and robust.

Some helpful YUI components are the Connection Manager, Layout Manager, TabbedView Layout and Animation.

4 Statement of the Problem

The project aims to tackle about classifying data using its context and apply it in a search engine framework. The system deals on how to deliver accurate and relevant list of medical records given an input medical record. The relevancy is based on the keywords extracted from the input record. The system must also summarize the result and outputs various statistical data that can be used to help plan a more effective treatment. Using XQuery in eXist, the project aims to grab its advantages for handling dynamic data to create a dynamic search engine capable of being deployed based on the domain preferences or context. Advantages pertaining to the flexibility of the data storage(XML) and the language (XQuery). Dynamic in the essence

of deployment of the system in any context where in this thesis, the medical record domain. This study also aims to open the concept of search engines to other fields aside from websites where the system can handle its domain.

The main target or audience of the project depends on the use of the system since the proposed method is supposed to be dynamic. However, this study focuses more on people who needs medical diagnosis, specially those who lacks access to modern medicine. The system can give them a second opinion on their diagnosis and plan a better treatment.

5 Methodology

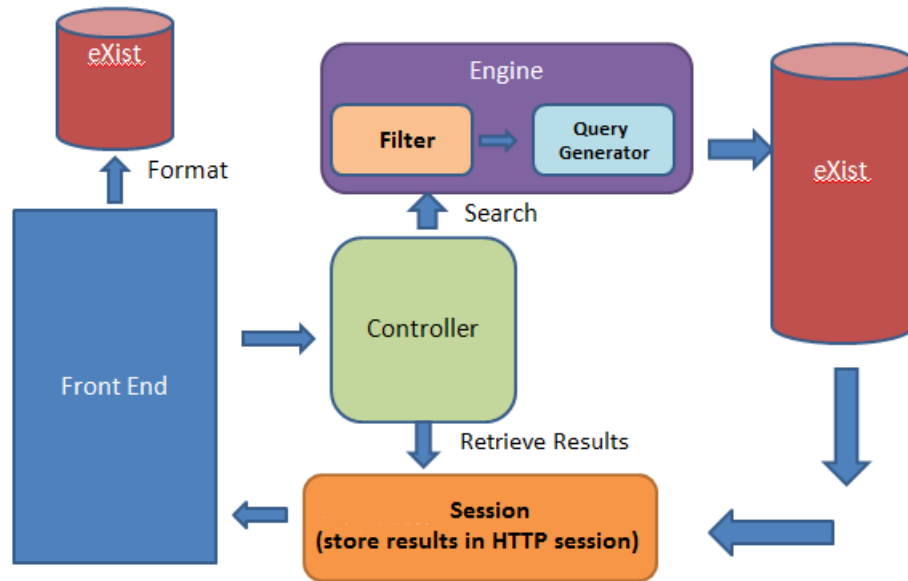


Figure 3: The MVC Framework of MedEx.

MedEx uses a Model-View-Controller architecture to be able to han-

dle its AJAX capability and be more dynamic. Once the user interacts with the interface (front end), it triggers the associated Javascript file to send a request to the controller, which forwards or redirects the requests to the appropriate handler. This uses eXist's module for URL rewriting and redirection: XQueryURLRewrite. Whenever a search is requested, the controller forwards this to the search engine, wherein the keywords are parsed and passed on to the eXist database. The database then passes the results to the HTTPsession for immediate storage. Once the search action finishes, the Javascript file communicates again with the controller to retrieve the data from the session, which is then passed back to the Javascript file which then changes the interface by populating a data table with the results.

Most of the system's code is in XQuery with XHTML and XML tags as its output. The view, model and controller are coded in the XQuery language with the help of eXist modules (recall XQueryURLrewrite) which made the system more dynamic and powerful. Part of the view is coded with Javascript for AJAX functionalities. The main advantage of using XQuery for the controller, as well as the view and model components, it gains full control of the system given that only XQuery is used from its front-end to the server portion, thus making the whole power of the language available for URL rewriting, an essential tool for passing requests to a given URL. eXist also introduces some advantages including Indexing and Scoring

The proposed method suggests four parts.

- Format Parsing
- Content/Data Classification and Filtering
- Search Engine
- Delivery

5.1 Format Parser

The program will accept an XML format file stored in the database. However, the contents of the file input may vary since the desired system must be dynamic. In this implementation, the SOAP format of medical records will be taken as input. Parts of the system need to access the format file in for it to function according to its configuration. This is used in the generation of input sections for the XHTML and parameter passing for the query generator.

The program parses the format file in order to generate input sections in the XHTML file. Given its record structure, the data in the format file contains configurations that the user wish to implement. Text Fields, Text Areas and even radio buttons are automatically created in the page depending on the format. The parameter-passing for the query generator also depends on the format since it requires the names of the input sections to access its value.

The sections of the format file contains "name", "display", "value",

"type", "reqd", "single", "input" and "summary". The tag "name" is for the name of the field (string value). The tag "display" tells the system to display it in the table result when the search execute (boolean value). The tag "value" which tells the system what boost value the text input have for the lucene indexing (float value). The tag "type" which tells the system what data type it is for the purpose of the range indexing ("int", "string", "boolean", "float", etc). The tag "reqd" which tells the system to check if that input field have a value in it, if not, it will require the user to put a value into it (boolean value). The tag "single" tells the system if the field have multiple values or not (boolean value). In this application, SOAP fields are multiple values since there are multiple symptoms, observations, assessment and plans. This is for the display of multiple values in the result table to separate values found in multiple tags (e.g. `plans_iplans_1i/plan_iplans_2i/plan_i/plans_i`). The tag "input" contains tags such as "tag", "size" and "element". The tag "tag" tells the system the input type for the said field ("none", "text", "text area", "radio box"). The tag "size" is for the tags "text" and "text area" of the "tag" element. This tag contains the element "row" and "column" which contains the values that pertains the length of the text field. The tag "element" of the input element contains the values for the radio buttons. The last format tag is the "summary" which contains configurations for the system which input fields in the results are to be summarized by the system. The "summary"

tag contains elements "summarize", "level", and "reference". The element "summarize" is just a boolean value that tells the system if the field is to be summarized. The element "level" has a value either 1 or 2. If its value is one, it means that the field will be summarized from the whole set of results. Level 2 means that it will be summarized from the level 1 results it is referenced to. And lastly the "reference" element may have multiple values pertaining to the fields its referenced to (string value).

To restrict the query to a certain collection in the database, there is also a root address added to the format file. This ensures that all queries done are from that collection to avoid conflict with similar collections. This is also to ensure that only one collection be used.

5.2 Content/Data Classification and Filtering

From the input sections, when the user submits the data for searching, the values from the input sections are passed as a parameter for the controller to process. It will be passed to the engine where parsing of parameters are done. In the engine, stopwords are stored. Stopwords are the words that are excluded from the keywords to be passed in the search engine. Examples of these words are articles which are "the", "to", "an". From here the filter now counts the number of appearance of a word or phrase then stores all distinct words and phrases in a sequence to be passed to the search engine.

The concept done here is compared to a puzzle where its divided into

puzzle pieces. In this implementation, the puzzle piece is a keyword extracted from the input file. The more puzzle piece found in the document, the more relevant the result files are.

5.3 Search Engine

The Content/Data Classification and Filtering is in a pipeline together with the Search Engine to make the data passing more easy. The engine will then access the sequence produced by the content classifier. Given these keywords or phrases, the search engine will generate a query based on the content. The query generator is based from the query of the module Lucene in Exist. This query from Lucene can store series of keywords and have the capability to make the query strict and ordered. Strict query ensures that all the keywords are found in the results. Ordered query ensures that keywords in the results are in the same order with the input file. This queries are combined in a function to have an easy configuration between 3 kinds of queries. These queries are strict-ordered, strict-unordered and Unstrict. This will then output results related to the input with appropriate ranking. Rankings are done through the ft:score function of the Lucene module. These results are records in the form of XML sequences where it is arranged according to the most relevant record. The engine will now store these results in the HTTP session to be accessed by the another function for display in the XHTML page.

In the search engine, eXist, an open - source native XML database will be used. A good thing about eXist is that it can already rank the results with the help of the Apache Lucene. Using its index configuration, the system can be configured to boost the scores of medical records using a built-in function *boost* (this has to be manually done though). Proper configuration of the index of all the collections has a huge influence on the results.

5.4 Delivery

During the loading of the page, the system will output an XHTML format stored in an XQL file as the main page of the system. This page contains all the necessary controls for the user where with the help of AJAX, the page only reloads at the start of the system. All other actions are incorporated with AJAX. Some tabs of the page are "Home", "Record Management", "Search", "Results", and "Help".

The tab "Record Management" is for members and admin accounts only. Its sole use is for "add", "edit", and "delete" records in the database. The admin accounts have an additional task which is to approve or disapprove pending records which are added, edited and deleted. This ensures that only the admin is able to modify the data in the database. For security issues, the admin filters the pending records to ensure that they are credible before approving them.

The tab "Search" is where the form for the search is located. It also

contains the search options "Strict" and "Ordered". This is where the user fills up the form for searching. The user have an option of Filling up the form or Importing a text file that contains the records. Import from text file have a certain format to follow to ensure proper importing of data. Once the search button is clicked, all the data in the form are passed to the controller via parameter.

The tab "Results" is where the results from the search are stored. This is stored in a table with helpful functions like paging of results, column sorting, column rearrangement and configuration for the number of results displayed. All results are arranged at default through their rank number. All the results are loaded at one time in the browser so that the page won't reload anymore if the user is browsing other pages of the results. You could click a row which is a medical record and a panel will slide from the left displaying the whole record of the clicked row. This gives the user easy access to the data stored in the record. Another panel from the right slides in and displays the summary of the whole results.

This contains the appropriate results and has the calculated statistical data. The delivery or the front end is coded with XHTML that is stored in an XQuery function. Underlying scripts for the AJAX framework is done with the help of YUI, additional CSS and Javascript, and XQuery.

| RANK | id | case_number | service | diagnoses | pathological_diagnoses | course | plan | outcome | SCORE |
|------|--------|-------------|-----------|--------------------------------------|------------------------|---|---|-----------|-------------|
| 1 | 5,614 | 3,295,604 | GS1 | ruptured appendicitis | final | S/P EL Appendectomy/GETA last May 1, 2008 | MGH Meds: Co-amoxiclav 750mg BID x 5 days Acetaminophen 120mg QID for pain | recovered | 5.4721264 |
| 2 | 21,045 | 3,431,914 | Pediatric | Ruptured Appendicitis | final | | | recovered | 0.111692995 |
| 3 | 6,928 | 3,297,920 | PediaSurg | ruptured appendicitis | final | | | improved | 0.10700975 |
| 4 | 15,340 | 3,380,409 | PediaSurg | Ruptured Appendicitis | final | | | improved | 0.09363954 |
| 5 | 16,037 | 3,363,574 | GS1 | ruptured appendicitis | final | | | recovered | 0.0895264 |
| 6 | 5,802 | 3,295,138 | GS3 | Ruptured appendicitis | final | | | improved | 0.06025732 |
| 7 | 14,855 | 0 | GS1 | ruptured appendicitis | final | | | recovered | 0.07381886 |
| 8 | 22,459 | 3,434,863 | GS1 | ruptured appendicitis | final | 5/16: Patient Admitted at Ward 2 Bed 35 | | recovered | 0.07339676 |
| 9 | 6,057 | 3,309,388 | GS1 | acute appendicitis, gangrenous acute | final | | | recovered | 0.07215677 |

Figure 4: Screenshot of the results page.

6 Results

The results of the study shown in appendix A are the given scores of relevancy and are ordered from the highest. Scores are computed by nodes in the given data. Each node may have a score between 0.0 to 1.0 based on its context from its node and its descendants. If matches are found in its descendants, all scores of the nodes are sum up to arrive at the final score. Based on the results, the more relevant keywords extracted from the input and matched to the database records, the more relevant it is. Since more keywords and phrases are found in a document, its context is near from the input, meaning more relevant. And also based from the results, the more data there are in the database, it is to assume that the most relevant data in the results is accurately relevant to the input data. Here we can say that

if more accurate data is to be stored in the database, the results are more accurately relevant.

6.1 Speed

To determine how much time it takes for querying, a Mozilla Firefox add-on called Firebug was used. Firebug allows one to edit, debug and monitor CSS, HTML and Javascript live in any webpage. Its console can display the amount of time it takes to complete a request using its JavaScript profiler. With Firebug, we were able to see the difference between using Lucene and Range together and Lucene alone for querying. The tests were done on single field queries and all field (11 fields) queries per 1000 records. Starting from 1000 records, the database was continuously increased by increments of 1000. Seen on the graph the the searching time slightly increases as the number of records was increased up to 22,697.

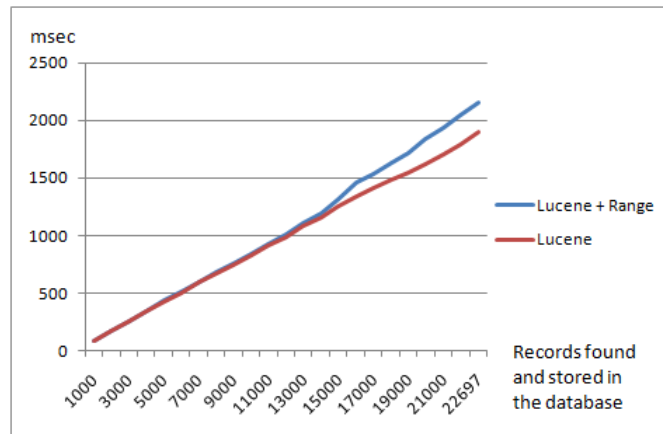


Figure 5: Searching time on single field.

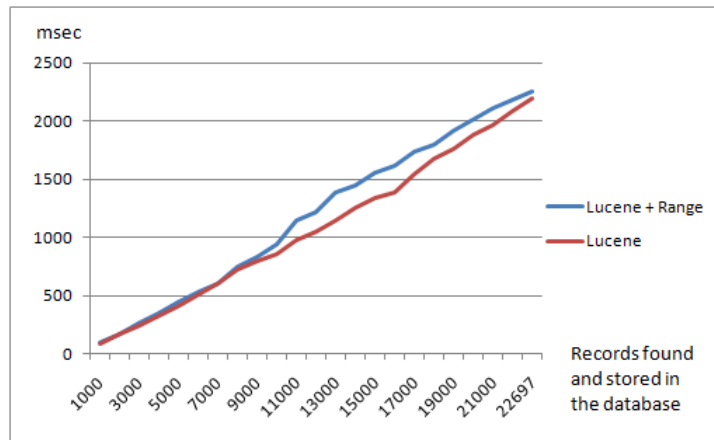


Figure 6: Searching time on all eleven fields.

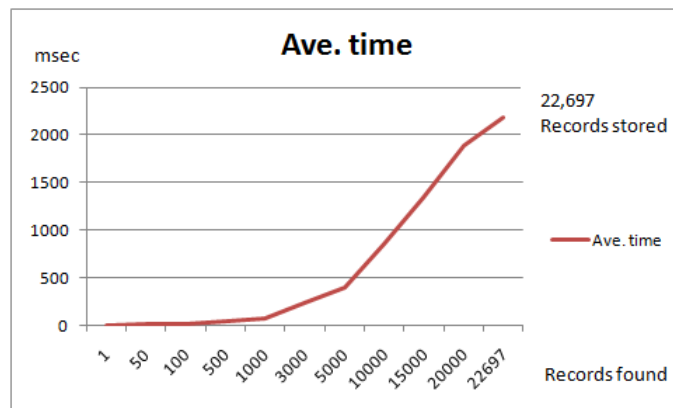


Figure 7: Average of the results.

Using Lucene Indexing, the query speed averages up to 2.0 seconds for the whole 22,697 records, which is considerably faster than using Lucene and Range Indexing which is 2.0-3.0 seconds. There was an option of using NGRAM for the query but was set aside in preference for Lucene.

As seen from the average speed table, results speed are determined by

how many results are queried. If all records are queried as shown in the figure 5 and figure 6, we can see that the speed forms a linear graph, ranging from 70-100 milliseconds per 1000 records. If the results return only a portion of the total records store, as shown in figure 7, we can see that the speed is proportional to the amount of results. If the search only found few results, the time amount the query to finish is very small. This results say that the more accurate your search is, less results are found. If less results are found, the search is faster.

6.2 Relevancy

A relevancy evaluation measures how reliable the system is when it comes to the ranking of results. Think of this as a simple question, simple answer test. How well does the search engine do when people submit the clearest search queries? The best results should appear at the top of the list.

For the relevancy of records, random records were removed from the database then searched. Resulting records were then checked for their relevancy from the removed record. Resulting records were proven similar or related to the removed record. In the said application which is the medical record domain, results found using the input file contains similar cases. In the top records, it showed that most of the records have the same diagnoses as the input record.

Relevancy of the system can be further improved if more records are

placed in the database since the system can reference more records.

7 Conclusion

Based from the results, the following conclusions are generated:

1. The dynamic feature of the system was made successfully with the help of XQuery using a format file to generate input sections and to pass format parameters.
2. The more relevant keywords and phrases extracted from the input, the results are more accurately relevant as shown in the results.
3. Using UMLS Metathesaurus generates a more accurate output since it helps the system understand synonymous terms and keywords.
4. The more records are stored in the database, the more accurate the results since more data will support the relevancy of the results.
5. Using the summary reports of each search, we can assume that the highest percentage of the same assessment is said to be the most relevant data to be crossed referenced.
6. It is indeed helpful by using the system as a supporting structure for search engines across other domains.

7.1 Contributions of the Study

1. Using XQuery as a the programming language greatly helps in developing dynamic systems since XML data is flexible and XQuery scripts can be embedded in XHTML pages to create sections that are based on a domain format.
2. Using eXist as the database is indeed helpful since data are stored in XML files that are either structured or unstructured. With the Apache Lucene, it greatly improved the performance of searching. User defined indexes also improved the performance of searching and with the help of boosting, it improves the relevancy since indexes with boost pertains to nodes that contains the most relevant parts.
3. The study reintroduced search engine ideas as a major tool for searching in other domains aside from websites. With this dynamic system, it may be a base structure of search engines in other domains aside from Medical Records.
4. Keywords in Context as implemented still contributes a large portion in the relevancy of the data.

7.2 Future Research

The researchers recommend the following for further study.

1. Using more algorithms to extract accurate keywords in the input data.

Some of the records in the that are not in the top relevant results may be more relevant, but since it's keywords are not found maybe because keywords found in these data are only related to the keywords extracted from the input.

2. Structuring and arranging of extracted keywords may help in determining its relevancy. Some context models are not based from its keywords but in its structure.
3. Using UMLS feature of serial coding may help in eliminating synonymity which helps in the 1st recommendation.
4. Integration with Open MRS since Open MRS makes it possible for one to export to an XML file. This can now be imported to the system to act as the primary source of records.
5. Have an installer for those who wish to use the system for developing programs and be deployed to a domain they want (e.g. school records, police records, forms)
6. Field Search Constraint instead of Whole Text Searching where instead of searching from the whole document. The search will be limited to the section where the keywords are expected to appear.
7. Index Creator where it is attached to the installer so that the user will not anymore make an index file for his records and files.

References

- [1] Soap notes. medical assistant net . advanced medical assistant custom web design. http://www.medicalassistant.net/soap_note.htm, 2009.
- [2] Unified medical language system. <http://www.nlm.nih.gov>, 29 January 2010.
- [3] Apache lucene. <http://lucene.apache.org/java/docs/>, March 2010.
- [4] Yui. <http://developer.yahoo.com/yui/>, March 2010.
- [5] exist. <http://www.exist-db.org>, November 2009.
- [6] R. Asleson and N. T. Schutta. *Foundations of Ajax*, 2005.
- [7] S. Amer-Yahia C. Botev and J. Shanmugasundaram. *A TeXQueryBased XML Full Text Search Engine*. SIGMOD2004, Paris, France, 2004.
- [8] I. Manolescu D. Florescu, D. Kossman. *Integrating Keyword Search Into XML Query Processing*. Elviesier Science BV, 2000.
- [9] A. Gaudinat et. Health search engine with e-document analysis for reliable search result. *International Journal of Medical Informatics*, 2006.
- [10] Jesse James. Garret. Ajax: A new approach to web applications. 2010.
- [11] Tom Negrino and Dori Smith. Javascript and ajax for the web, sixth edition. 6th ed. 2007.

A Deployment

A.1 Requirements for Deployment

1. At least 256MB RAM (Preferably 1GB or more)
2. Javascript supported browser (Preferably Firefox v3.6)
3. Operating System that supports Java
4. Java Run-Time Environment
5. Exist (Native XML Database) ver. 1.4.0 rev. 10440
6. Yui 2 module
7. Search Engine Application (e.g. MedEx)

A.2 Installation for Deployment: (Windows)

1. Install Firefox (download at www.mozilla.com) [Figure 8]
2. Install JRE for Java Support (download at www.sun.com/java)
3. Install Exist Native XML Database (download at www.exist-db.org)
[Figure 9]
4. Install Exist as Service (located at the start menu - Exist) [Figure 10]
5. Start eXist as a service [Figure 11]
6. Open browser and go to the url <http://localhost:8080/exist> [Figure 12]

7. Go to Admin (located in the Left Tab) [Figure 13]
8. Log in using your admin account and password [Figure 14]
9. Go to install tools [Figure 15]
10. Install both DB Admin Application and XQuery Sandbox [Figure 16]
11. After installation, Go to the Home Folder of exist - webapp - Paste Deployment Folder here [Figure 17]
12. Visit your Application Page (<http://localhost:8080/exist/FolderName>)

A.3 Search Engine Application Modification

1. Create a Format File for record format [Figure 18]
 - The format file is composed of sections where each section is an input field in the record (e.g. Name, Age)
 - Each Section is composed of Elements namely:
 - (a) name - name of the Field (string)
 - (b) display - if the field is going to be displayed in the results table (boolean)
 - (c) value - the value of the boost for the Lucene index (float)
 - (d) type - the data type of the field for the Range index (string - "int", "string", etc)
 - (e) reqd - if the field is required to be filled up by the user. (boolean)

- (f) single - if the field contains single or multiple values. This is for multiple values in an element - this is for creation of delimiters in the result table. (boolean)
- (g) input - what is the input type in the form. Contains:
 - i. tag - Input type (e.g. text_area, text, radio_box)
 - ii. size - if input type is text_area or text, the elements inside this element is <row> and <column> which contains the integer size. Else if the input type is a radio_box, the element/s inside this element is <element> which contains the choices in the radio buttons.
- (h) summary - how is the field summarized in the summary window.
 - i. summarize - set to true if the field is to be summarized after querying, false otherwise.
 - ii. level - if level 1, it is summarized based on the results. Else if level 2, it is summarized based on the referenced field. (int - 1,2)
 - iii. reference - the referenced field (string). Its value is always a level 1 field. For sections with a level 1 summary, this is left blank.

2. Create a User File for user authentication [Figure 19]

The file contains the information for each user in the application.

The file contains of <user> tags that is composed of:

- (a) username - the username of the user (string)
- (b) password - the password of that account (string)
- (c) account - the membership type of the account (string)

3. Edit the medex.xml for collection query restriction of the application
(example: "/db/pgh") [Figure 20]
4. Create an Index File for the Record based from the format (example:
Medical Record) [Figure 21]
5. Open the Exist client in the start menu then log in [Figure 22].
6. In the home("/db"), create a collection according to the name in the
edited medex.xml ("/db/CollectionName") [Figure 24]
7. After creating the collection [Figure 25], create 3 collections inside
that collection namely:
 - (a) format
 - (b) record
 - (c) user
8. Go back to home ("/db") then go to system->config->db. Inside this
collection, repeat step 6 and 7.

9. In the system->config->db->CollectionName->format, store the index file for the format that is found in your Application Folder->DB files->format.xconf [Figure 26].
10. In the system->config->db->CollectionName->user, store the index file for the user that is found in your Application Folder->DB files->user.xconf [Figure 27].
11. In the system->config->db->CollectionName->record, store the index file that you created for the records. Example of this file is located in the Application Folder->DB files->record.xconf [Figure 28].
12. Go back to the home collection ("/db"). Go to CollectionName -> format and import the format.xml you have created [Figure 29].
13. Next is going to the CollectionName->user and import the user.xml you have created [Figure 30].
14. Next is going to the CollectionName->record and store all the records in xml format [Figure 31].
15. After storing all the data, you can now use the whole application.

A.4 Tuning Exist

A.4.1 Memory Settings

By default, Java limits the amount of memory that it can access, thus eXist will not automatically exploit the machines available memory. Tuning the memory can optimize searches on large databases. After installing eXist as a service, users may want to edit `EXIST_HOME/tools/wrapper/conf/wrapper.conf` to adjust Java Heap Size to their preference. Recommended setting is at least 512 MB of memory.

A.4.2 Cache Settings

All of the core database files and indexes has a page cache. It is used to load frequently used pages in the db faster. If the cache is too small, eXist may unload pages then reload them a few moments later, which is called "trashing effect". Users may want to tune eXist to have a larger cache size. This can be done by editing `EXIST_HOME/conf.xml`. Recommended setting is at least 128M of memory.

A.5 Using the Application (MedEx Deployment)

1. Tabs

- (a) Home - contains the information of the Application [Figure 32]
- (b) Record Management - new, edit, delete records. This can only be accessed by the admin and members only [Figure 33]

- i. New
 - ii. Edit
 - iii. Delete
 - (c) Search - contains the form for searching [Figure 34]
 - i. Text Input
 - ii. Import File
 - (d) Results - contains a table where results are stored [Figure 35]
 - i. Column Sorting
 - ii. Column Editing
 - iii. Page
 - iv. Row Click
 - (e) About - contains information about the creators of the program.
2. Record View [Figure 36]
- contains the whole information of a record clicked
3. Summary View [Figure 37]
- contains the summary of the whole results found.

B Screenshots



Figure 8: Firefox Installation.



Figure 9: Exist Installation.

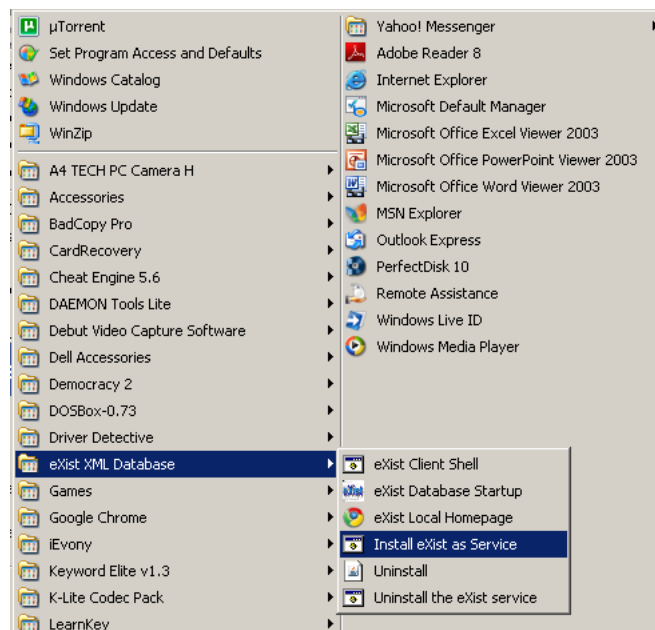


Figure 10: eXist as a service.

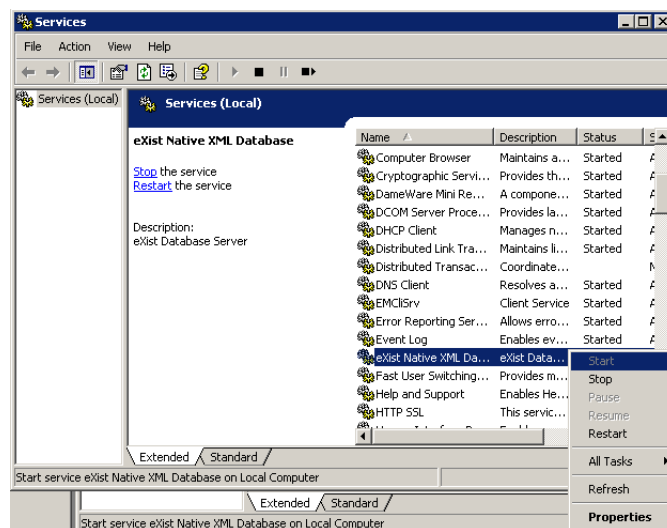


Figure 11: Start eXist as service.

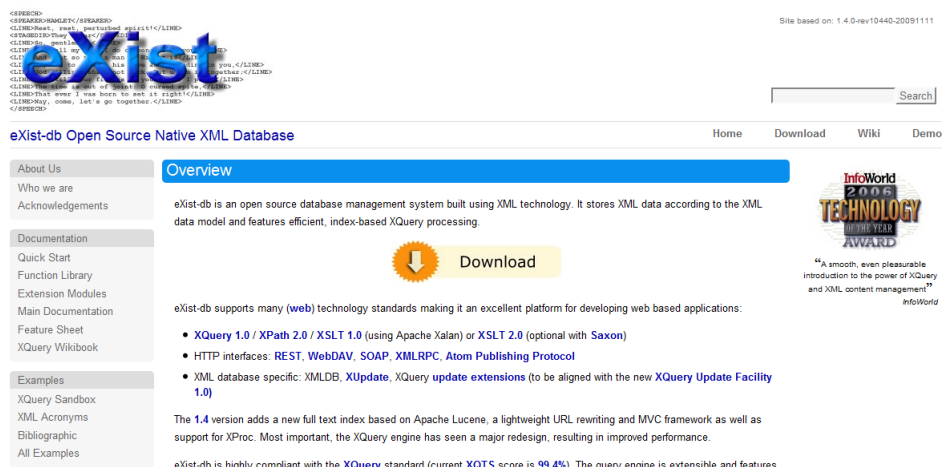


Figure 12: eXist home page.

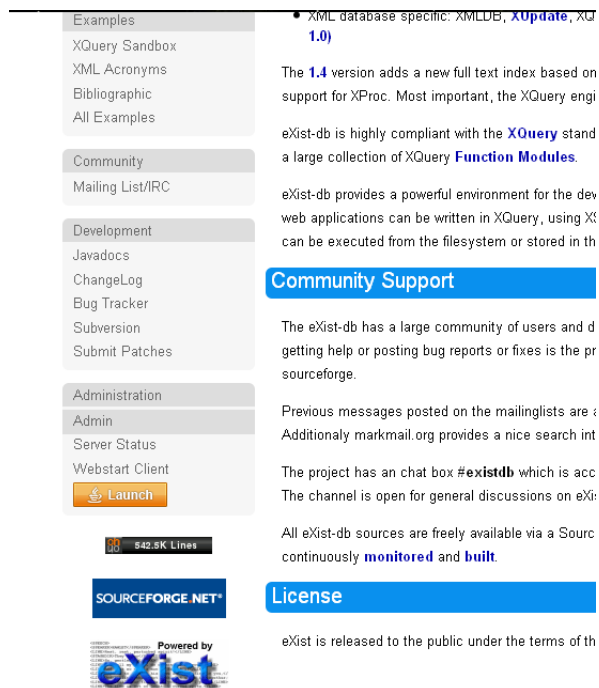


Figure 13: Select admin.

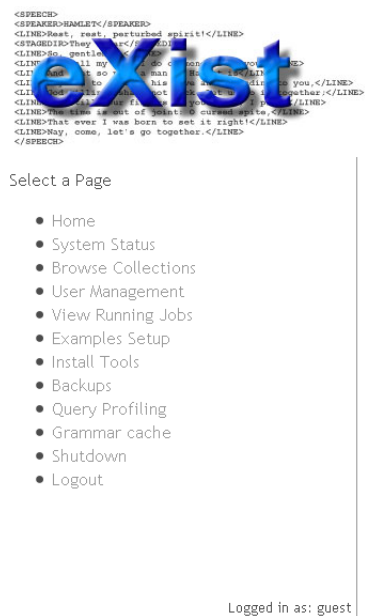
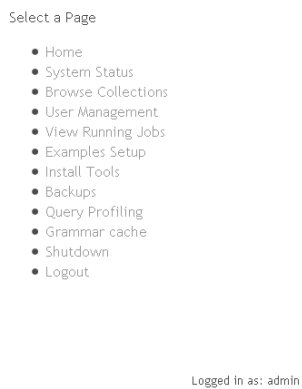


Figure 14: eXist login page.



Figure 15: Select install tools.



Use this module to install selected XQuery tool apps into the database. The tools will be stored into the db collection `/db/www`.

Modules:

- Install

The screenshot shows a Windows XP desktop with a blue taskbar. The active window is 'webapp' (Internet Explorer). The Explorer window displays the 'webapp' folder contents. The left sidebar has 'File and Folder Tasks' and 'Other Places' sections. The 'MedEx' folder is selected, and a tooltip shows its details. The address bar indicates the path 'C:\Program Files\exist\webapp'.

File and Folder Tasks

- Rename this folder
- Move this folder
- Copy this folder
- Publish this folder to the Web
- Share this folder
- E-mail this folder's Files
- Delete this folder

Other Places

- exist
- My Documents
- Shared Documents
- My Computer
- My Network Places

Details

MedEx
File Folder
Date Modified: Sunday, April

Address Bar: C:\Program Files\exist\webapp

Folder Contents:

- admin
- api
- apps
- cocoon
- functions
- gsoc
- irclog
- MedEx (selected)
- resources
- sandbox
- scripts
- styles
- stylesheets
- WEB-INF
- xforms
- xproc
- xqls
- xquery
- acknowledge
- ant-tasks
- atompub
- backup
- building
- client
- cluster
- configuration
- controller.xql
- credits
- debugger
- deployment
- developer
- devguide
- devguide_c...
- devguide_in...
- devguide Jo...
- devguide_rest
- devguide_s...
- devguide_...
- devguide_x...
- devguide_x...
- dln_tree
- documentation

MedEx Folder Details:

- Size: 132 MB
- Folders: controller, Records, scripts, view, yui
- Files: collection.xconf, controller.xql, engine.xql, ...

Status Bar: 1 objects selected

44

```

1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <format>
3   <Section>
4     <name>id</name>
5     <display>true</display>
6     <value>1</value>
7     <type>int</type>
8     <reqd>false</reqd>
9     <single>true</single>
10    <input>
11      <tag>none</tag>
12    </input>
13    <summary>
14      <summarize>false</summarize>
15    </summary>
16  </Section>
17  <Section>
18    <name>case_number</name>
19    <display>true</display>
20    <value>1</value>
21    <type>int</type>
22    <reqd>false</reqd>
23    <single>true</single>
24    <input>
25      <tag>none</tag>
26    </input>
27    <summary>
28      <summarize>false</summarize>
29    </summary>
30  </Section>
31  <Section>
32    <name>service</name>
33    <display>true</display>
34    <value>1</value>
35    <type>string</type>
36    <reqd>false</reqd>
37    <single>true</single>

```

Figure 18: Format file.

```

1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <users>
3   <user>
4     <username>Administrator</username>
5     <password>adminpassword</password>
6     <account>admin</account>
7   </user>
8   <user>
9     <username>derekstiles</username>
10    <password>healingtouch</password>
11    <account>member</account>
12  </user>
13  <user>
14    <username>angriethompson</username>
15    <password>internationalnursinglicense</password>
16    <account>member</account>
17  </user>
18  <user>
19    <username>gregkalif</username>
20    <password>caduceuspresident</password>
21    <account>member</account>
22  </user>
23  <user>
24    <username>hoffman</username>
25    <password>mastersurgeon</password>
26    <account>admin</account>
27  </user>
28  <user>
29    <username>taylorchase</username>
30    <password>slicketsharp</password>
31    <account>member</account>
32  </user>

```

Figure 19: Users file.


```

1 xquery version "1.0";
2
3 (:~
4   Displays the main page of the application.
5   This is also where the main collection that will be used is stored.
6   A function generates all the fields based from the format given
7 :)
8
9 declare namespace MedEx="http://localhost:8080/exist/MedEx";
10 declare namespace system="http://exist-db.org/xquery/system";
11 import module namespace util="http://exist-db.org/xquery/util";
12 import module namespace request="http://exist-db.org/xquery/request";
13 import module namespace session="http://exist-db.org/xquery/session";
14 import module namespace xdb="http://exist-db.org/xquery/xaldb";
15
16 declare option exist:serialize "method=xml media-type=text/xml omit-xml-declaration=no indent=no";
17
18 (:~
19   Generates all the fields in the form depending on the format given.
20   It supports text fields, text areas and radio buttons
21 :)
22 declare function MedEx:get-inputs($form-number as xs:int?) as element(){
23
24   let $null := session:set-attribute('collection', "/db/pgh")
25   let $sequence := collection(session:get-attribute('collection'))//Format/Section
26   let $null := session:set-attribute("format", $sequence)
27   return
28     <form action="#" id="form($form-number)" name="myform($form-number)">
29       <div>
30         (if($form-number eq 2) then
31           <div>

```

Figure 20: Editing medex.xql.

```

1 <collection xmlns="http://exist-db.org/collection-config/1.0">
2   <index xmlns:atom="http://www.w3.org/2005/Atom"
3     xmlns:html="http://www.w3.org/1999/xhtml"
4     xmlns:wiki="http://exist-db.org/xquery/wiki">
5
6     <fulltext default="none" attributes="no"/>
7     <!--Lucene Indexes-->
8     <Lucene>
9       <analyzer class="org.apache.lucene.analysis.standard.StandardAnalyzer"/>
10      <analyzer id="us" class="org.apache.lucene.analysis.WhitespaceAnalyzer"/>
11      <text qname="name"/>
12      <text qname="display"/>
13      <text qname="value"/>
14      <text qname="type"/>
15      <text qname="reqd"/>
16      <text qname="single"/>
17      <text qname="tag"/>
18      <text qname="row"/>
19      <text qname="column"/>
20      <text qname="element"/>
21    </Lucene>
22    <!--Range Indexes-->
23    <create qname="name" type="xs:string"/>
24    <create qname="display" type="xs:boolean"/>
25    <create qname="value" type="xs:int"/>
26    <create qname="type" type="xs:string"/>
27    <create qname="reqd" type="xs:boolean"/>
28    <create qname="single" type="xs:boolean"/>
29    <create qname="tag" type="xs:string"/>
30    <create qname="row" type="xs:int"/>
31    <create qname="column" type="xs:int"/>
32    <create qname="element" type="xs:string"/>
33  </index>
34  <triggers>
35    <trigger event="store,remove,update" class="org.exist.versioning.VersioningTrigger"/>
36  </triggers>
37 </collection>
38

```

Figure 21: Creating index file.

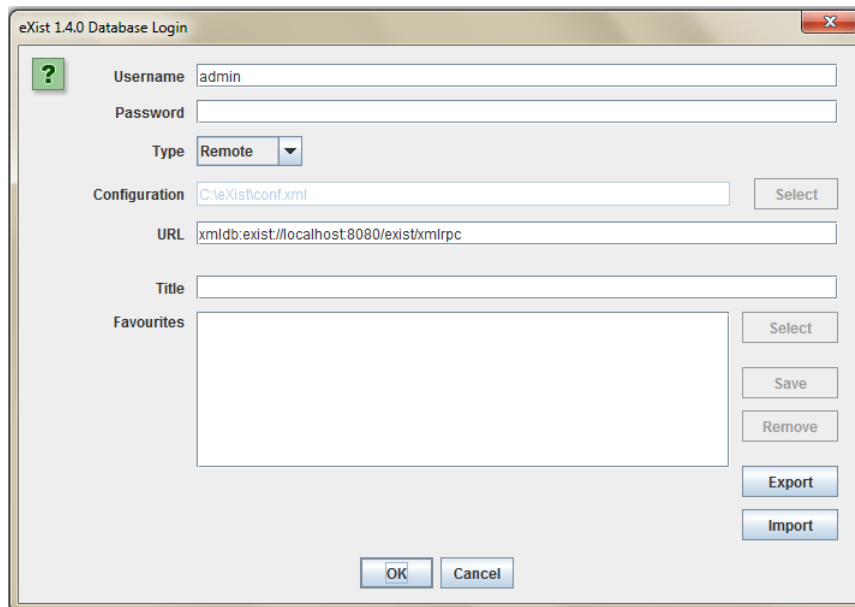


Figure 22: eXist db login page.

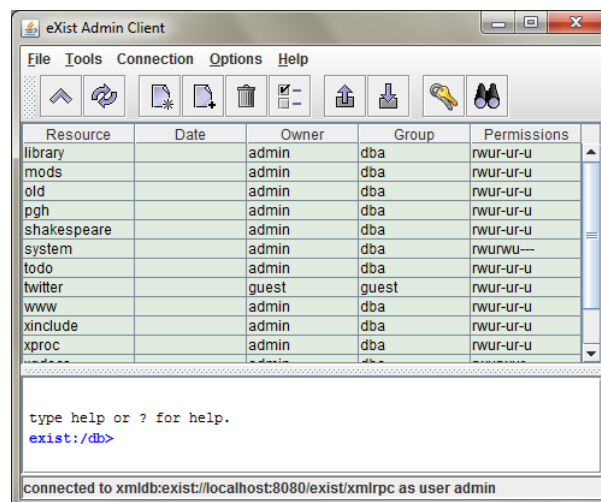


Figure 23: eXist db home collection.

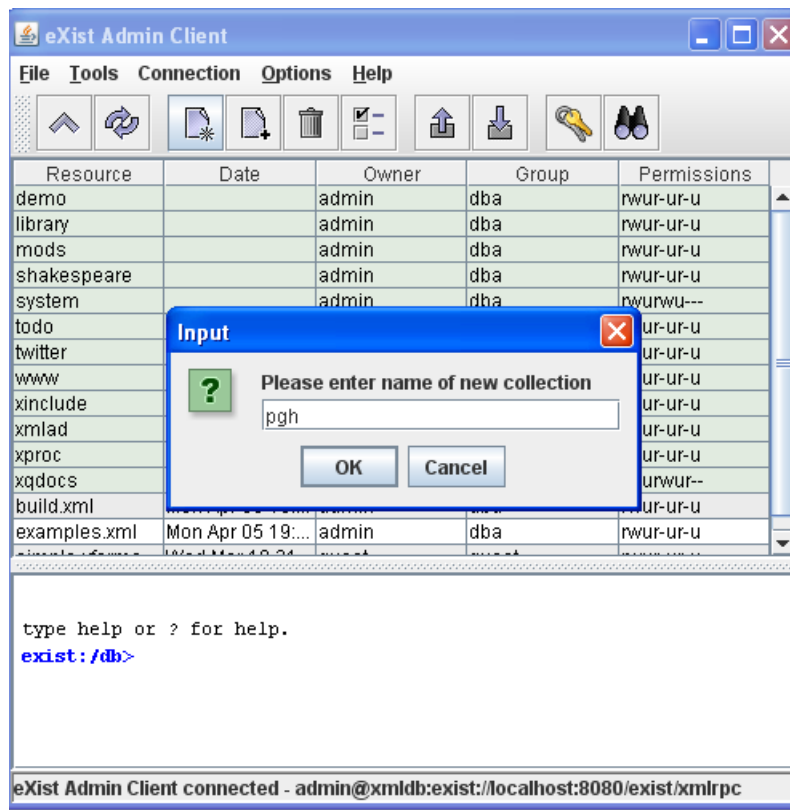


Figure 24: createcollection.

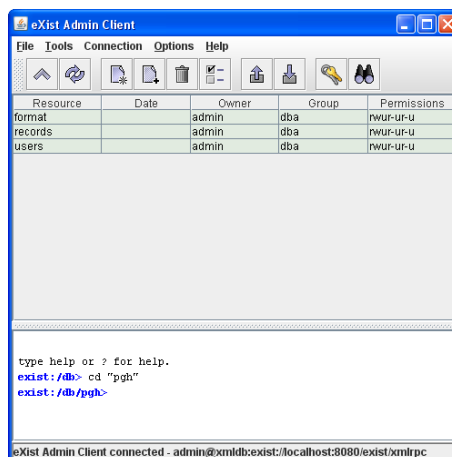


Figure 25: eXist db domain (PGH) collection.

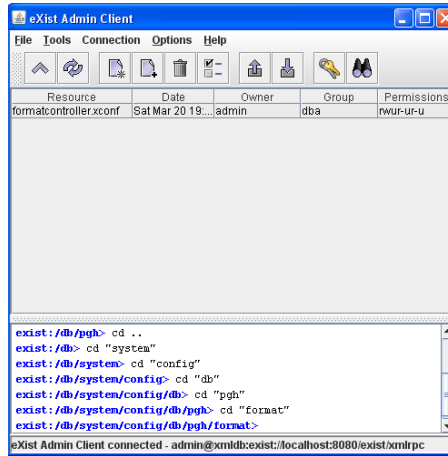


Figure 26: eXist db format index collection.

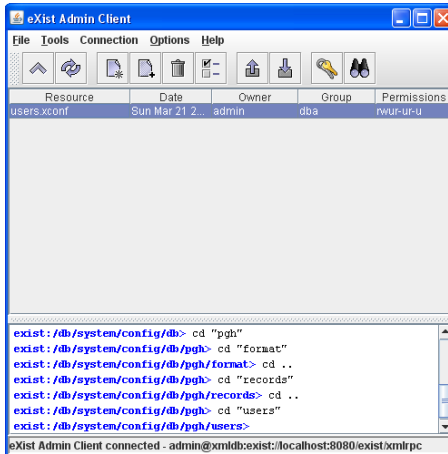


Figure 27: eXist db users index collection.

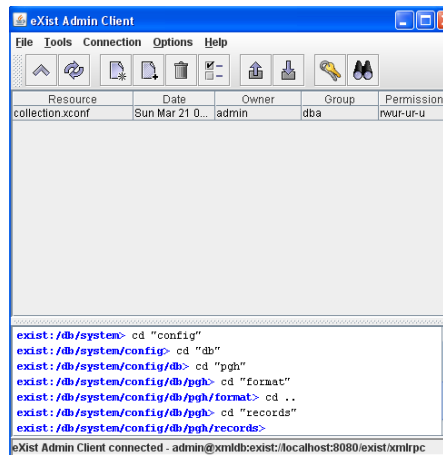


Figure 28: eXist db records index collection.

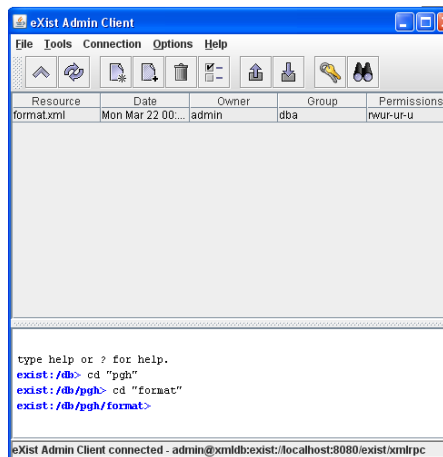


Figure 29: eXist db format collection.

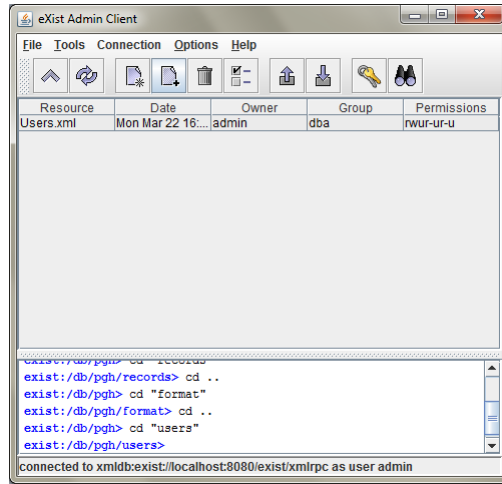


Figure 30: eXist db users collection.

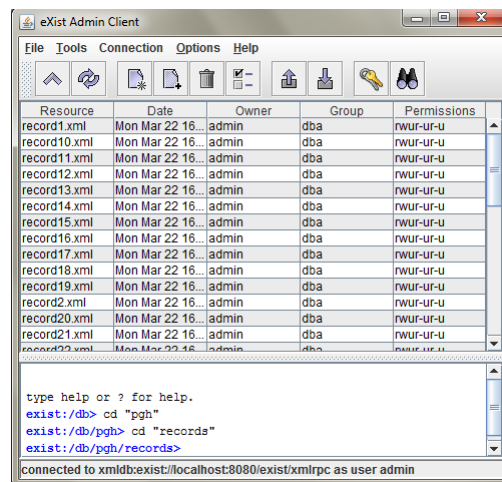


Figure 31: eXist db records collection.



Figure 32: MedEx home page.



Figure 33: Record Management.

MedEx: Medical Cross-Referencing System

Logged in as guest: [Log in](#)

[Home](#) [Search](#) [Result](#) [About](#)

Search for Related Records

[Submit Record](#) [Reset Record](#)

Text Input

Input from File [Browse...](#) [OK](#)

☒ Select Search ☐ Ordered Search

[Search](#) [Reset](#)

service:

complaint:

history:
1 day PTC patient noted periumbilical pain gradually shifting to the right lower quadrant. This was associated with anorexia and vomiting

physical examination:
(+) guarding on all quadrants (+) direct tenderness on right lower quadrant (+) direct and rebound tenderness on left lower quadrant

diagnoses:
ruptured appendicitis

Copyright © Da Kyfers

Figure 34: Search form.

MedEx: Medical Cross-Referencing System

Logged in as guest: [Log in](#)

Results Retrieved:
Found 22697 records in 0 seconds.

[Home](#) [Search](#) [Result](#) [About](#)

<< first < prev 1 2 3 4 5 next > last >> 25

| RANK | id | case_number | service | diagnoses | pathological_diagnoses | course | plan | outcome |
|------|--------|-------------|------------|-----------------------|------------------------|---|--|-----------|
| 1 | 5,614 | 3,295,604 | GS1 | ruptured appendicitis | final | S/P EL Appendectomy/GETA last May 1, 2000 | MGH Meds: Co-amoxiclav 750mg BID x 5 days Alcorne 120mg OD for pain | recovered |
| 2 | 21,045 | 3,431,914 | Pediatrics | Ruptured Appendicitis | final | | | recovered |
| 3 | 6,928 | 3,297,920 | PediaSurg | ruptured appendicitis | final | | | improved |
| 4 | 15,340 | 3,380,409 | PediaSurg | Ruptured Appendicitis | final | | | improved |
| 5 | 16,037 | 3,383,574 | GS1 | ruptured appendicitis | final | | | recovered |
| 6 | 5,802 | 3,295,138 | GS3 | Ruptured appendicitis | final | | | improved |
| 7 | 14,855 | 0 | GS1 | ruptured appendicitis | final | | | recovered |
| 8 | 22,459 | 3,434,863 | GS1 | ruptured appendicitis | final | 5/16: Patient Admitted at Ward 2 Bed 35 | | recovered |

[Click to sort ascending](#)

Record

RESULT NUMBER: 1
SCORE: 5.4721264
id: 5614
case number: 3295604
service: GS1
complaint: right lower quadrant pain
history: 1 day PTC patient noted periumbilical pain gradually shifting to the right lower quadrant. This was associated with anorexia and vomiting
physical examination: (+) guarding on all quadrants (+) direct tenderness on right lower quadrant (+) direct and rebound tenderness on left lower quadrant
diagnoses: ruptured appendicitis
pathological diagnoses: final
course: S/P EL Appendectomy/GETA last May 1, 2000
operation, histopath result: A. Acute suppurative appendicitis with periappendicitis B. Acute inflammatory pattern. Lim
plan: MGH Meds: Co-amoxiclav 750mg BID x 5 days Arcoxia 120mg OD for pain
outcome: recovered

Figure 35: Results page.

| Record |
|---|
| RESULT NUMBER: 1 |
| SCORE: 5.4721284 |
| id: 5614 |
| case_number: 3295604 |
| service: GS1 |
| complaint: right lower quadrant pain |
| history: 1 day PTC patient noted periumbilical pain gradually shifting to the right lower quadrant. This was associated with anorexia and vomiting. |
| physical examination: (+) guarding on all quadrants (+) direct tenderness on right lower quadrant (+) direct and rebound tenderness on left lower quadrant |
| diagnoses: ruptured appendicitis |
| pathological diagnoses: final |
| course: S/P EL Appendectomy/GETA last May 1, 2008 |
| operation finding: ruptured appendix at tip, adherent to retroperitoneal wall, exudative suppuration |
| operation_histopath_result: A. Acute suppurative appendicitis with penappendicitis B. Acute inflammatory pattern. Lim |
| plan: MGH Meds: Co-amoxiclav 750mg BID x 5 days Arcoxia 120mg OD for pain |
| outcome: recovered |

Figure 36: Record view.

| Summary |
|--|
| AGE: |
| 19 : 46.153847% |
| 20 : 15.384615% |
| 50 : 7.6923075% |
| 22 : 7.6923075% |
| 47 : 7.6923075% |
| 23 : 7.6923075% |
| 29 : 7.6923075% |
| GENDER: |
| male : 100% |
| SYMPTOMS: |
| fever : 16.071428% |
| nosebleeds : 1.7857143% |
| loss of appetite : 1.7857143% |
| iredness : 1.7857143% |
| jaundice : 1.7857143% |
| R Flank Pain : 1.7857143% |
| edema : 1.7857143% |
| exertional dyspnea : 1.7857143% |
| chestpain : 1.7857143% |
| generalized aches and pains : 14.285714% |
| cough : 12.5% |
| papitations : 1.7857143% |
| hematochezia : 1.7857143% |
| body weakening : 1.7857143% |
| poor appetite : 1.7857143% |
| headaches : 1.7857143% |
| lethargy : 1.7857143% |
| diarrhea : 1.7857143% |
| sore throat : 1.7857143% |
| stuffy nose : 1.7857143% |
| headache : 1.7857143% |

Figure 37: Summary view.