Content-Based Searching with Relevance Ranking for Learning Objects

By

Vikram Chellappa

Bachelor of Engineering (Electronics and Communication Engineering),

University Of Madras, Madras, India, 2001

Submitted to the Department of Electrical Engineering and Computer Science and the Faculty of Graduate School of the University of Kansas in impartial fulfillment of the requirements for the degree of Master of Science.

Dr. Susan Gauch, Chair

Dr. Jerry James, Committee Member

Dr. Perry Alexander, Committee Member

Date Project Accepted

Acknowledgments

It is a pleasure to acknowledge my people for their precious support, which has made this project possible.

I am deeply indebted to **Dr. Susan Gauch**, my graduate advisor and committee chair, for giving me this opportunity and guiding me through this research project. She has been instrumental in enabling me to conceive, design and execute every step involved in the making of this project. Every single time, she has overwhelmed me with technical inputs and ideas and has stood for me at all times. She has made this project a reality. I thank God for bringing me in association with a hard working person such as her. I shall cherish this association all through my life.

I would also like to thank Dr. Jerry James and Dr. Perry Alexander for serving as members on my committee. Special thanks to Dr. Jerry James for his constant support, help and encouragement during the spring 2003 semester. I am also thankful to Abishek, and Dinesh for their help and inputs at essential times.

I thank my parents, my brother, all my family members and friends for their care, affection, support and patience. I thank God for showering his blessings on all my good people and me.

Abstract

There is always a need for more efficient and accurate content-based searches. Numerous methods have been designed to develop these search engines. Another important requirement is to produce an ordering in the returned result set that matches the users needs. This is also being worked on and multiple algorithms have been formulated to achieve this. The common of all these methods is the vector-space model. The similarity between the document and the query is calculated using the cosine similarity formula where terms are weighted as the product of the Term Frequency and the Inverse Document Frequency.

The document list is sorted based on similarity values and the most important document is displayed first and the least important document is displayed last. Our project is designed to incorporate a vector space ranking algorithm for the content fields of XML documents.

Table of Contents

1. Introduction

Search engines are often the first method used by any user to locate a page. The requirements that a search engine should satisfy are that the search engine must be easy to use, help the user to find useful information, display results in a meaningful way, should have descent response time and help authors to improve the site. There are other numerous requirements to be taken into account but most of those considerations are based on other considerations like the scale of the project, the audience etc. In order to be a widely used search engine, it should satisfy all the above requirements without fail.

Most search engines work with HTML documents in which the tags can be largely ignored. XML documents have semantically meaningful tags. Lumping all the values in all the fields together for search purposes is a simple way to index them. So they are searchable by textual search engines.

In contrast, XML documents can be stored in relational Database management systems and searchable via SQL. These however have only rudimentary capabilities for searching for words within a textual field.

A final option is to store the XML documents in a native XML database. This approach is slower at search time, but it is more flexible than a Relational Database Management system since not all XML documents must confirm to the same schema. This latter approach is the one that is adopted by Intelligent Knowledge Management Environment (IKME) project at the University of Kansas.

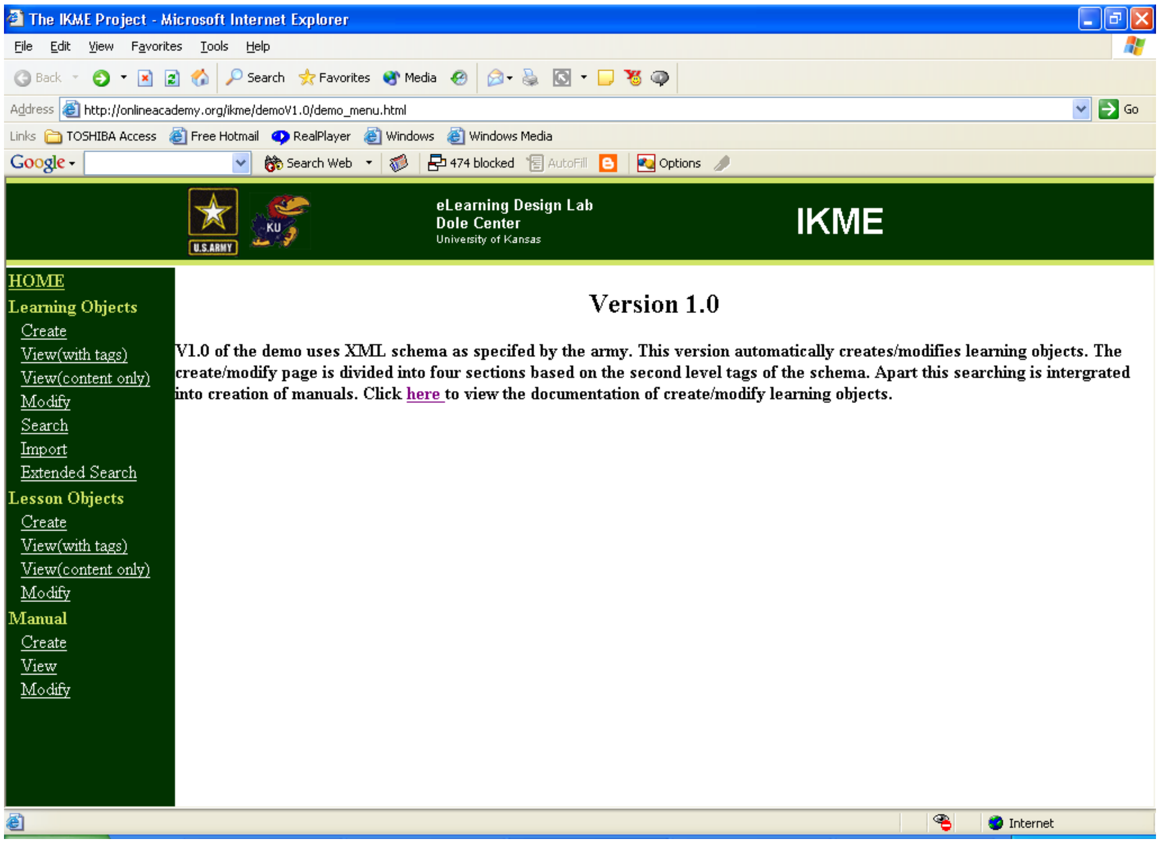
The IKME project stores XML documents in the eXist Database. eXist provides search capabilities using XPath queries however, when searching on the content field, the results come back in an alphabetical order. The goals of this project were to design and implement an efficient and suitable search feature for the content fields in the XML documents and generating a rank ordered result set with a fast response time.

2. IKME

Intelligent Knowledge Management Environment (*IKME*) is an ongoing project at the *University of Kansas* aimed at assisting the *Defense Information Technology Testbed (DITT)/University After Next (UAN)* by providing an advanced reach-back capability for commanders, staff, and other users who have time-critical needs.

In this project, Extensible Markup Language is used as the medium for representing data. This allows explicit separation of style and content. The *XML* schema is developed by the "*Center for Army Lessons Learned*". Knowledge creators use the environment to create learning objects that are stored as *XML* documents, and the same XML document can be represented in various styles and data formats using style sheets. The learning objects are in turn used to create lesson objects or manuals.

Reusability, interoperability, and extensibility are the major aspects of content development techniques. In this context, learning objects is a core concept. Reusable learning objects represent an alternative approach to content development. Learning objects are used by educators to break content into bit size chunks. These chunks could be independently created, maintained, reused, pulled apart and joined together. These chunks are called the Objects. This is the fundamental idea behind learning objects. The aim is to reduce the amount of effort and time necessary while incorporating modifications to the content and these learning objects highly reduce the time involved in content processing thus enabling the creator to achieve the desired goals at a much faster rate than other conventional techniques.


The IKME Project - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://onlineacademy.org/ikme/demoV1.0/demo_menu.html
Go
Links TOSHIBA Access Free Hotmail RealPlayer Windows Windows Media
Google Search Web 474 blocked AutoFill Options
U.S.ARMY
KU
eLearning Design Lab
Dole Center
University of Kansas
IKME
HOME
Learning Objects
Create
View(with tags)
View(content only)
Modify
Search
Import
Extended Search
Lesson Objects
Create
View(with tags)
View(content only)
Modify
Manual
Create
View
Modify
Internet

Version 1.0

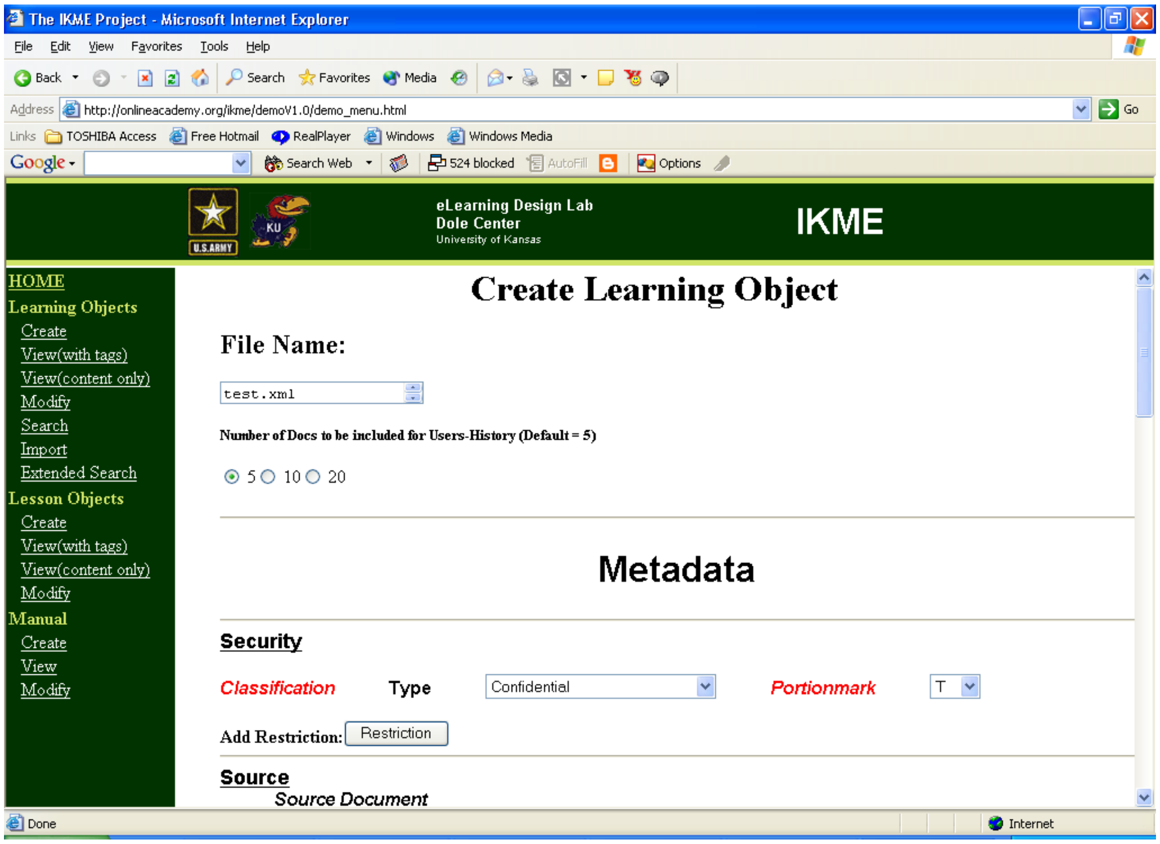
V1.0 of the demo uses XML schema as specifed by the army. This version automatically creates/modifies learning objects. The create/modify page is divided into four sections based on the second level tags of the schema. Apart this searching is intergrated into creation of manuals. Click here to view the documentation of create/modify learning objects.

Figure 2.1 IKME main page

2.1 Learning Objects

Learning objects are the fundamental content objects. They can be used independently or grouped into larger collections of content in order to be reusable in multiple contexts for multiple purposes, they must be tagged with metadata. This metadata is descriptive information that allows them to be easily found in a search.

A document’s content must be structured and separated from presentation information. This goal is accomplished by XML. XML uses tags to structure information and a separate XSL document is used to present information. The most straightforward method to display information is the conjoining of XML file with related style sheets defined in XSL.


The IKME Project - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://onlineacademy.org/ikme/demoV1.0/demo_menu.html
Go
Links TOSHIBA Access Free Hotmail RealPlayer Windows Windows Media
Google Search Web 524 blocked AutoFill Options
U.S.ARMY
KU
eLearning Design Lab
Dole Center
University of Kansas
IKME
HOME
Learning Objects
Create
View(with tags)
View(content only)
Modify
Search
Import
Extended Search
Lesson Objects
Create
View(with tags)
View(content only)
Modify
Manual
Create
View
Modify
Restriction
Done
Internet

Create Learning Object

File Name: test.xml

Number of Docs to be included for Users-History (Default = 5)

[✓ 5] 10 20

Metadata

Security

Classification Type Confidential Portionmark T

Add Restriction:

Source

Source Document

Figure2.2 Create Learning Object

IKME provides users the ability to create, view, modify or search learning objects by providing information on an online form and saving the learning object. These learning objects are stored as XML documents in the eXist database. These learning objects can be reused multiple times in different learning contexts by creating lesson objects or manuals.

2.2 Lesson Objects

The learning objects as described earlier can be grouped or aggregated to create lesson objects. It is the choice of the user to decide which learning objects could be grouped in order to form a lesson object. Here too, the user has the option to modify lessons objects.

2.3 Manuals

Learning objects can also be combined to generate manuals. Manuals are created for a particular topic and the user can select specific learning objects related to a particular topic. In this way, the user can create a manual for a particular topic. An important issue concerning the creation of manuals is the identification of learning objects that are related to the same topic. As the database grows, it gets harder to identify related learning objects. So, we need a method to automatically identify similar learning objects, thereby making it easier to create lesson objects and manuals.

2.4 eXist Database

eXist is an open source native XML Database which is written in Java and provides efficient index-based XPath query processing and extensions for keyword search. IKME uses this eXist database to store all the XML documents.

Documents are managed in hierarchical collections just as in a file system. The collections are not restricted to a particular schema and so an unlimited number of documents of multiple document types could be stored in the same collection. Conventional XPath processors are mostly based on top-down or bottom-up approach and even though they are clean approaches they become inefficient for large documents. This is because, in order to access any element, every single path from the top or the bottom must be traversed. Approaches differ but they all become inefficient for large document collections.

That is why indexed structures are essential in order to efficiently perform queries on large unconstrained collections. A numbering scheme is used to index in eXist. This enables faster identification of structural relationships like parent-child, ancestor-descendent or previous-next siblings between nodes. eXist provides index-based queries on full text-content of nodes by extending XPath. Additional operators for keyword search, i.e., wildcards, regular expression matches, and term proximity are the main extensions.

eXist comes with XUpdate support. This is a standard proposed by XML:DB initiative for updates of selected parts of a document. The database is integrated with Cocoon which is based on XML:DB API. The resources in eXist can be directly referenced from Cocoons site map with XML:DB pseudo-protocol. The development of dynamic web pages are achieved with an XML:DB based XSP logic sheet.

2.5 XPath

XML Path Language is a non-XML language for addressing parts of an XML document. XPath is syntax to define parts of an XML document. XPath defines standard library functions and is an integral element of XSLT. XPath allows the user to write expressions that could refer to any part of a XML document. XPath expression can also represent numbers, strings or Booleans. The expressions look very similar to expressions in a computer file system. XPath 2.0 is an expression language that allows the processing of values conforming to a specific data model. The data model provides a tree representation of XML documents as well as atomic values such as integers, strings, and booleans, and sequences that may contain both references to nodes in an XML document and atomic values. The result of an XPath expression may be a selection of nodes from the input documents, or an atomic value, or any sequence allowed by the data model.

2.6 XUpdate

XUpdate is a XML Update Language. The main purpose of this language is to provide open and flexible update facilities in order to modify data in XML documents. XUpdate uses XPath extensively to choose elements for updating and for conditional processing. The XUpdate is expressed as a well-formed XML document. This allows the user to create elements, attributes, text, processing instructions and comments. Other important functions are append, update, remove and rename.

2.7 XML:DB API

The XML:DB API is designed to enable a common access mechanism to XML databases. This enables the constructions of applications to store, retrieve, modify and query data stored in an XML database. Another important goal of this API is to be modular and define a simple baseline for users to follow. This gives the user the capability to manage, view and update the content of a native XML database. It allows the user to insert, update, and delete documents interactively. Lets the user add and delete collection of documents. The user could import and export documents from/to the file system. The user can search for data using XPath expressions, with support for namespaces as well. Modify the content of documents using XUpdate expressions. Export XPath query results to file. Support for DB username and password.

3. Goals

IKME offers a search feature that allows users to search for keywords and displays the documents in which the keyword appears. The main goal of this project is to provide a complete text search where in the user could search on any keyword and any number of key words and a rank ordered result set is produced. The next important goal is to have an extremely good response time. The querying, processing and displaying of the final result set, that is rank ordered is all done on the fly, i.e., all of these processes are done after the user enters a query. These processes have to be done with a quick response time. The other important goal was to be display the most accurate result set. In summary, shell the complete text search on the XML documents has to be accurate, adaptable, and extensible while providing quick response time.

4. Implementation

This project has been programmed in PERL and the implementation is completely in accordance to the norms set by the existing project. The eXist database as discussed in chapter 2, is a native XML database that stores all the XML documents. The major parts of this project are querying the database, manipulating the returned data, processing the query, matching the query with the content, and displaying the results. The main aim of the project is to obtain a rank ordered result set for any query or any number of queries the user enters. Currently, IKME provides a search feature that does not return a rank ordered result set. Rather, the results returned in response to a query are in alphabetical order. This project returns a result set ordered by similarity values. This results in a more accurate result set as will be shown by suitable examples and results.

4.1 Design Considerations

We had two options in implementing a vector space model for the content field.

We could either
- Write a full text search engine code in C++.
- Develop a post process/wrapper on eXist.

We chose the second option since we have eXist that stores all the documents. eXist also provides efficient index-based XPath query processing and extensions for keyword search.

4.2 Block Diagram

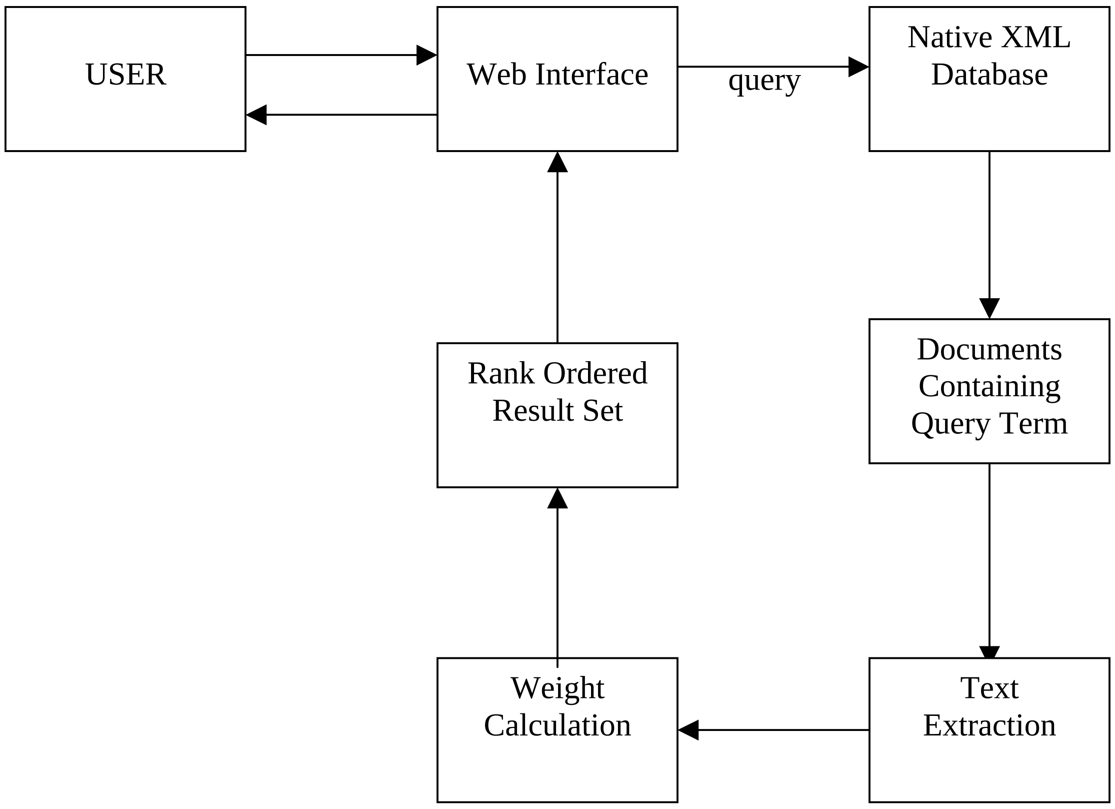


Figure 4.1 Block Diagram

- The user initially enters an input query string through a web interface.
- This input query string is processed (down cased, punctuation removal, checked for double quotes, tokenized).
- An XPath query is formulated. This is used to query the Native XML Database. The XML documents containing the input query term are obtained.
- The content tags in these documents are searched for. The text from the content tag from each document is extracted. The text is then tokenized.

- The independent terms in the input query string are compared with the independent tokens in each of the matching documents and the total number of matches for each query term is found.
- The similarity between the document and the query is calculated using the cosine similarity formula where terms are weighted as the product of the Term Frequency and the Inverse Document Frequency.
- A rank-ordered result set is then generated and displayed to the user through the web interface.

4.3 Query Processing

The first step we do is to process the input query string entered by the user. The input query string is checked for double quotes. If the quotes are seen at the beginning and the end of the query string the query is accepted as entered by the user, and no processing is done on them. The query is processed only if the query string is not within double quotes, indicating that the user does not require exact phrase matching.

If the input query string is not within double quotes, we first eliminate trailing and leading spaces from the input query string. The query is then scanned for punctuations. If there are any, then they are all removed. Now, the input query string is plain text. The query is then down-cased. After this the query is tokenized. The input query string is tokenized if there is more than one term in the input query string. The query is split into individual words just as the text from all the XML documents were done. These individual terms are the ones that are used to compare with the tokenized words from the text in each document to find matches.

4.4 Querying the Database

As we have seen, eXist is the native XML database used to store all the documents. A database may have unlimited set of documents and collections. There are two major functions ‘document()’ and ‘collection()’. Wild card characters could also be used as parameters to the two commands. The ‘document()’ function accepts single documents path and a list of document paths. The ‘collection()’ function specifies the collection whose documents are to be included in query evaluation. So, for our requirement the ‘collection()’ function has been used to specify the collection of documents and formulate the XPath query.

When the user has entered no query, the XPath query that has to be sent to the database, should browse the appropriate directory and retrieve all the documents. So the XPath query is made to search on the title element. This is the XPath query.

```
my $xpath_query = "collection('/db/ikme/Demo/')/object[*]//header/title";
```

When the user has entered an input query string the database is queried for only the documents in the collection containing the input query term in the content tag. Here the XPath query is the following piece of code.

```
$test = "'".$token."'";
my $xpath_query = "collection(\'/db/ikme/Demo/\')//content[contains(.,$test)]";
```

This following statement instantiates the RPC XML client with the address of the server. $token is the term in the input query string.

```
my $nxd_client = RPC::XML::Client->new('http://localhost:8081/');
```

The response from the database is collected, by sending the query to the native XML database RPC server. The following piece of code does this for us.

```
my $response = $nxd_client->send_request('query',$xpath_query,"ISO-8859-1",100,1,1);
```

After some error checking on the response the XML response got back from the RPC server is retrieved.

4.5 Using the XML: Twig Parser

XML:Twig is a PERL module that is a subclass of the XML-Parser to allow easy processing of XML documents of all sizes. The XML:TWIG parser offers a tree-oriented interface to a XML document. XML:TWIG is extremely flexible. It allows the user to dump parts of the tree, set call-backs during processing, on tags and sub-trees or process only a part of the tree.

We use the XML:Twig parser to process huge documents. This offers the tree-based processing model and gives the user the capability to make the decision, of how much of the tree has to be loaded at once in the memory. XML:TWIG is also completely PERL based. The other parsers are XML:Parser, XML:SIMPLE or XML:DOM. But in all these parsers it is either tough to formulate code or it is very complex to handle large documents or these parsers are not very powerful.

Here a new parser is instantiated to read the output from the RPC server. The XML:Twig parser is used. The retrieved result is then parsed using the new instantiated parser.

```
my $result_id = $response->value;
my $twig_parser = XML::Twig->new();
$twig_parser->parse($result_id);
```

These three lines of code indicate the retrieval of the response got back from the server, the instantiation of the new XML Parser, and the retrieved result is parsed using the XML:Twig parser respectively.

After the retrieved result has been parsed, the root method offered by XML:TWIG is used to obtain the root of the twig.

```
my $root=$twig_parser->root;
```

Now, the attribute values are obtained by the att method. The attributes obtained are the hit count and the query time.

```
$number_of_docs = $root->att('hitCount');
$query_time = $root->att('queryTime');
```

The children method is used to obtain the titles.

```
my @children = $root->children('title');
```

Subsequently, the filenames are also identified using this method. Here we obtain an array of titles and another array that contains the filenames. The two arrays have the same indexes and the filenames are corresponding to the titles. So, every element in the array containing titles has a corresponding element in the other array containing filenames. The text is extracted from the content tags and stored in another array. Thus we build three parallel arrays, one each for titles, filenames, and contents.

4.6 Formula

Let us now look at the formulae used. The first parameter calculated is the term frequency (TF). The term frequency is the measure of how often a term is found in a particular document.

TF_{ij} = frequency of $term_i$ in doc_j / number of words in doc_j

Each term in the input query string, is compared with the tokens of the text in every document containing the query term, to find the number of matches. We use this and the number of words in each document, i.e., the document length that we have calculated earlier to find the relative term frequency.

The next parameter we calculate is the Inverse Document Frequency (IDF).

IDF_i = log (\$j/\$documents_containing_term$_i$)

\$j = total number of documents in the collection

\$documents_containing_term$_i$= number of documents in which the term occurs

The number of documents in which the term occurs divides the total number of documents in the collection. The log of this value gives the Inverse Document Frequency.

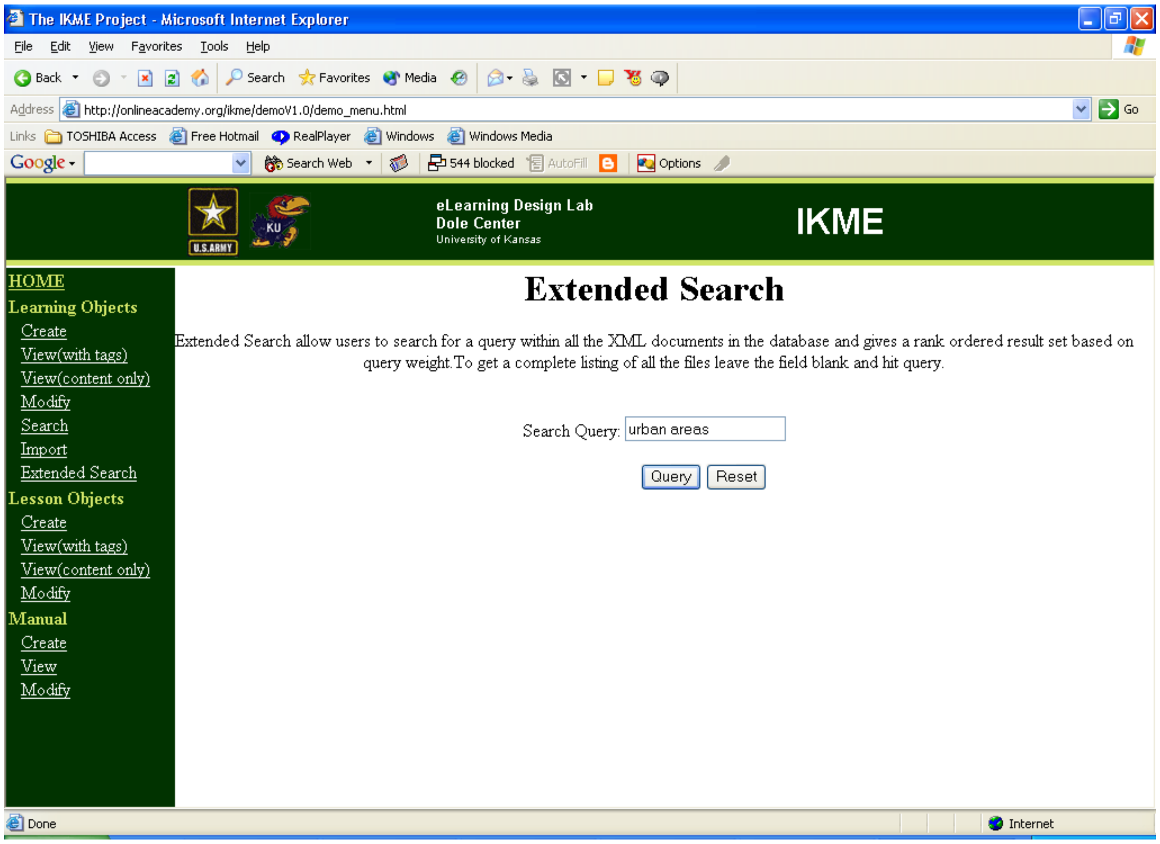
Finally, we calculate the weight of the term in each document. The weight identifies the importance of the query term in each document. So, the higher the weight of a term in a document the more important is that document and vice versa.

Weight$_{ij}$ = Term Frequency$_{ij}$ * Inverse Document Frequency$_j$

We store the weights in another array, i.e, we store the weights in an accumulator. If there is just one term in the query string, then the process ends here. We have calculated the weight of the query term in every document that has this term. So this weight array is sorted in descending order.

If there is more than one term in the query string then we have to do one step more. The whole process of querying the database for matching documents, calculation the Term Frequency, the Inverse Document Frequency and the weight is done for the next term in the input query string. This weight is added to the weight of the previous query term. This step is repeated for each term in the query string until all the terms are exhausted.

So, now we have to sort the array having the weights, in descending order. This will be the final order of the documents. The most important document will be displayed first in the order and the least important document will be displayed last in the order.



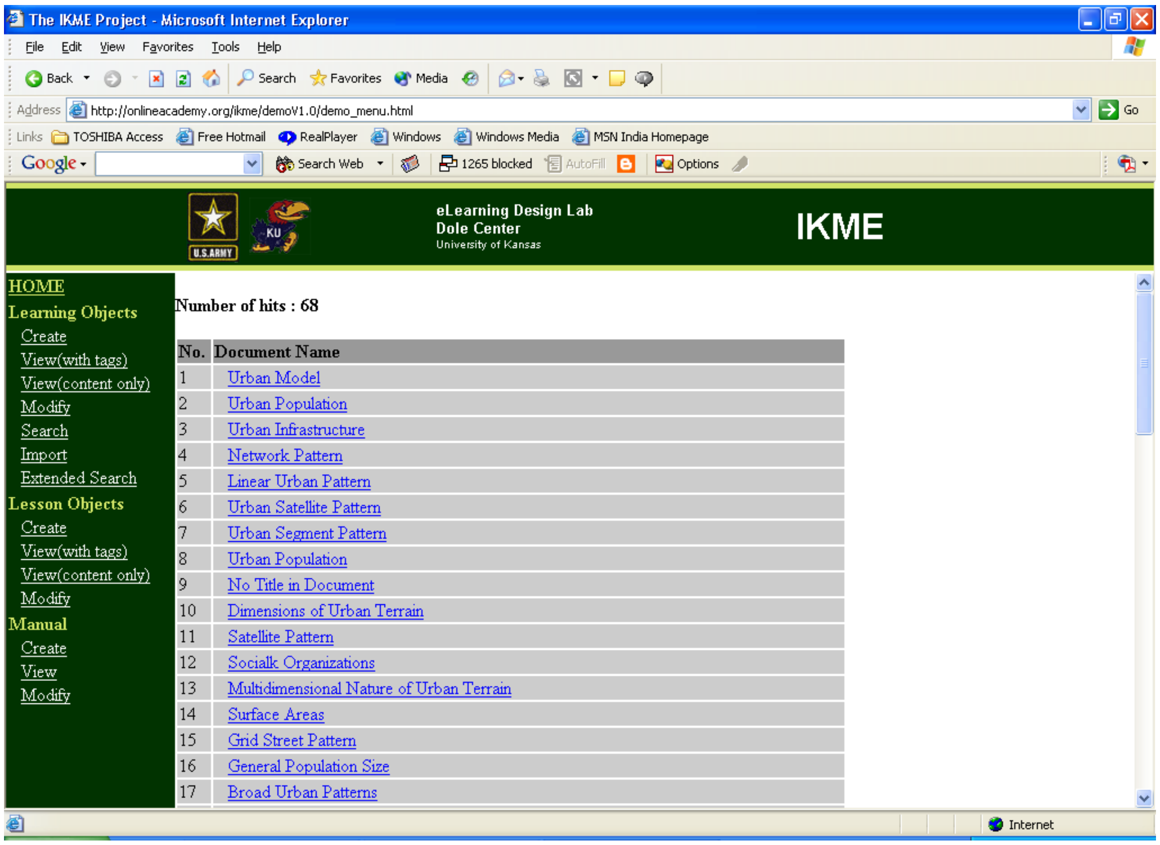
Extended Search

Extended Search allow users to search for a query within all the XML documents in the database and gives a rank ordered result set based on query weight.To get a complete listing of all the files leave the field blank and hit query.

Search Query: urban areas

Figure 4.2 Search Query

The above figure shows the extended search page. Here, the user can enter an input query string and obtain the rank ordered result set. The rank-ordered result set is shown in the next screen shot.



Number of hits : 68

No.	Document Name
1	Urban Model
2	Urban Population
3	Urban Infrastructure
4	Network Pattern
5	Linear Urban Pattern
6	Urban Satellite Pattern
7	Urban Segment Pattern
8	Urban Population
9	No Title in Document
10	Dimensions of Urban Terrain
11	Satellite Pattern
12	Socialk Organizations
13	Multidimensional Nature of Urban Terrain
14	Surface Areas
15	Grid Street Pattern
16	General Population Size
17	Broad Urban Patterns

Figure 4.3 Rank Ordered Result Set

Figure 4.3 shows the rank ordered result set for the query string in figure 4.2. These documents are rank ordered. The first N documents have both ‘urban’ and ‘areas’.

The next M documents have only ‘urban’ occurring X times.

The next K documents have only ‘areas’ occurring Y times.

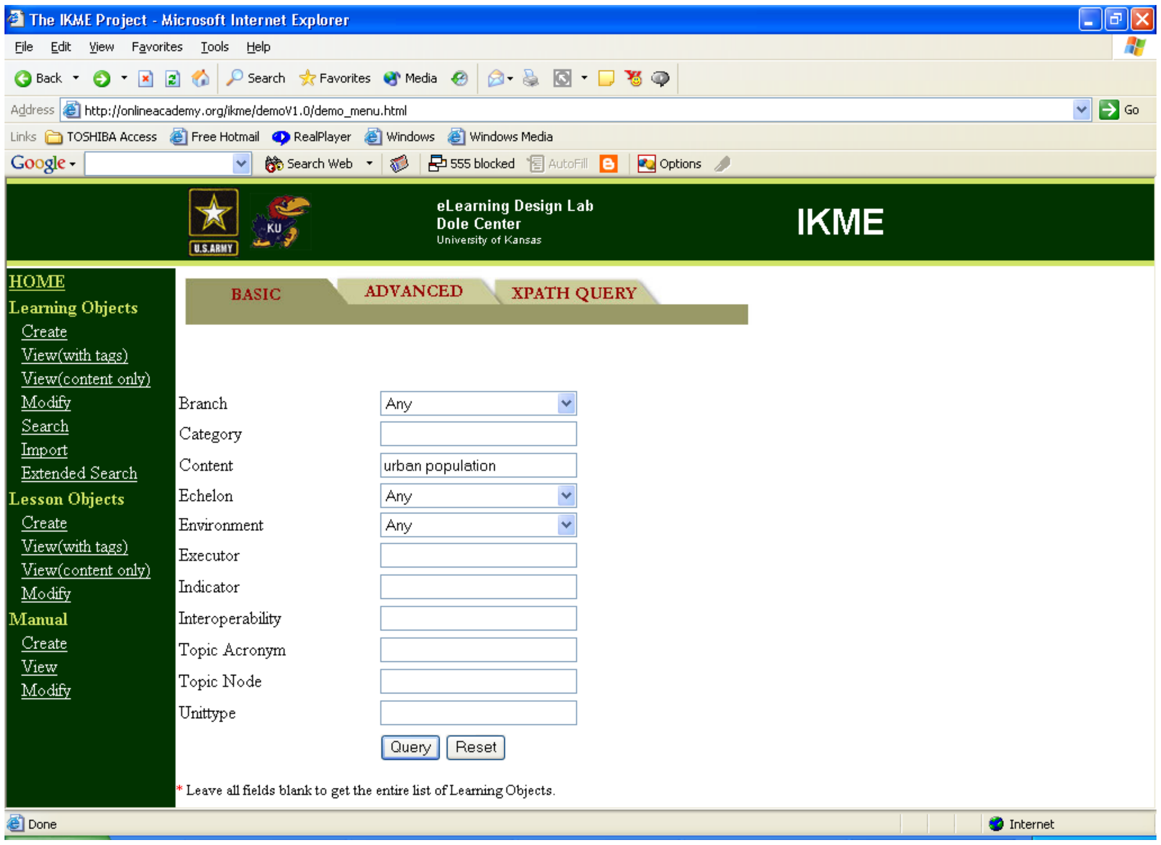
Where X is greater than Y.

The lowest weighted documents have only a single occurrence of either ‘urban’ or ‘areas’.

5 Features and Improvements

Let us discuss the features, advantages, and improvements achieved in this project when compared to the already existing search feature provided by eXist.

- This project offers a complete text based search engine. The content tag of every XML document in the collection is scanned and the input query string is compared with the tokens and matches are found.
- This project yields rank ordered result set. The weight of the input query string is calculated for each document and the weight is directly proportional to the importance of the document for the input query string.
- The query within quotes is taken as it is and the query string is compared with the tokenized text in all documents for matches.
- All the punctuations in the input query string are removed. So the result set for a query string without punctuations and a query string with punctuations will have the same rank ordered result set.
- When there is more than one term in a query string, the already existing search finds only those documents in which there are all the terms in the input query string. This extended search displays documents which contain all the query terms or any one of the terms or a few of them, since the search is implemented by calculating the weight and so the user gets the best and the most accurate result set. This can be seen in the following screen shots.

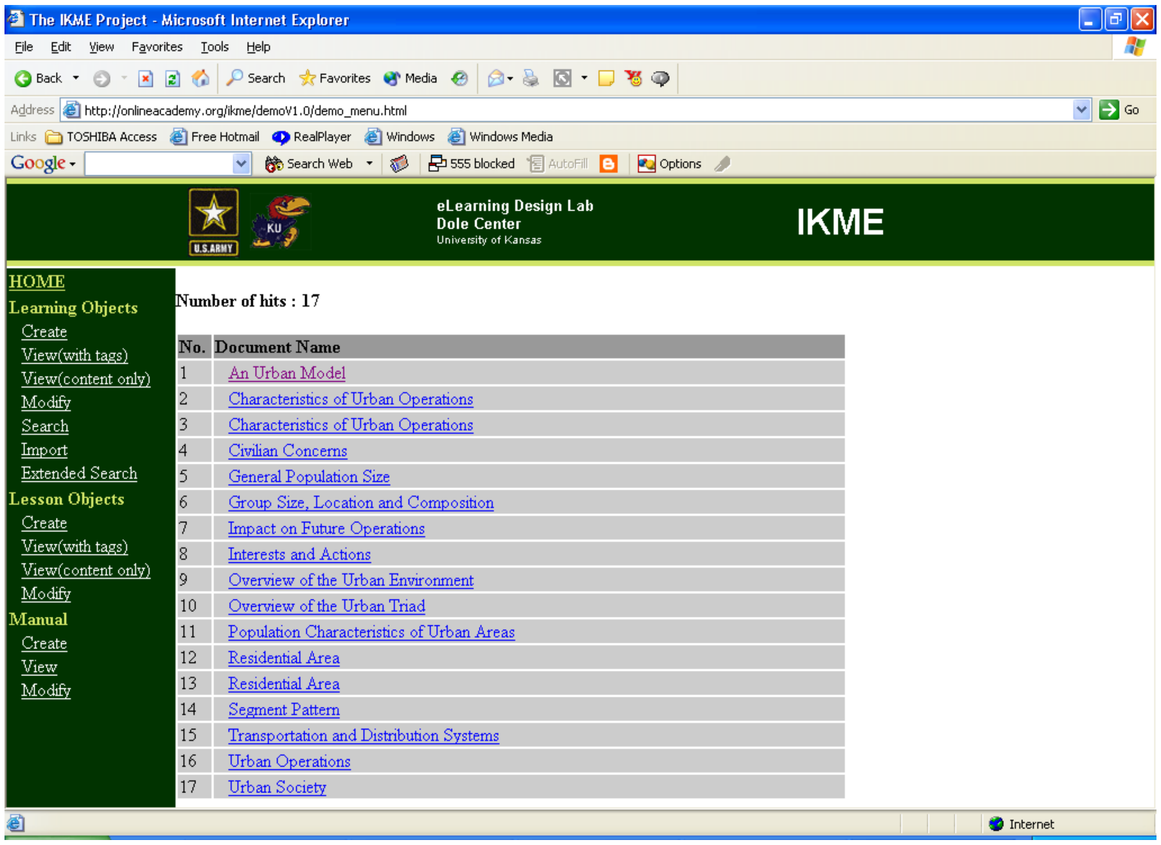


Branch	Any
Category	
Content	urban population
Echelon	Any
Environment	Any
Executor	
Indicator	
Interoperability	
Topic Acronym	
Topic Node	
Unittype	

* Leave all fields blank to get the entire list of Learning Objects.

Figure 5.1 Search page with query

This is the already existing search. The following screen shot shows the result set for the query ‘urban population’.



The IKME Project - Microsoft Internet Explorer

http://onlineacademy.org/ikme/demoV1.0/demo_menu.html

eLearning Design Lab
Dole Center
University of Kansas

IKME

HOME
Learning Objects
- Create
- View(with tags)
- View(content only)
- Modify
- Search
- Import
- Extended Search

Lesson Objects
- Create
- View(with tags)
- View(content only)
- Modify

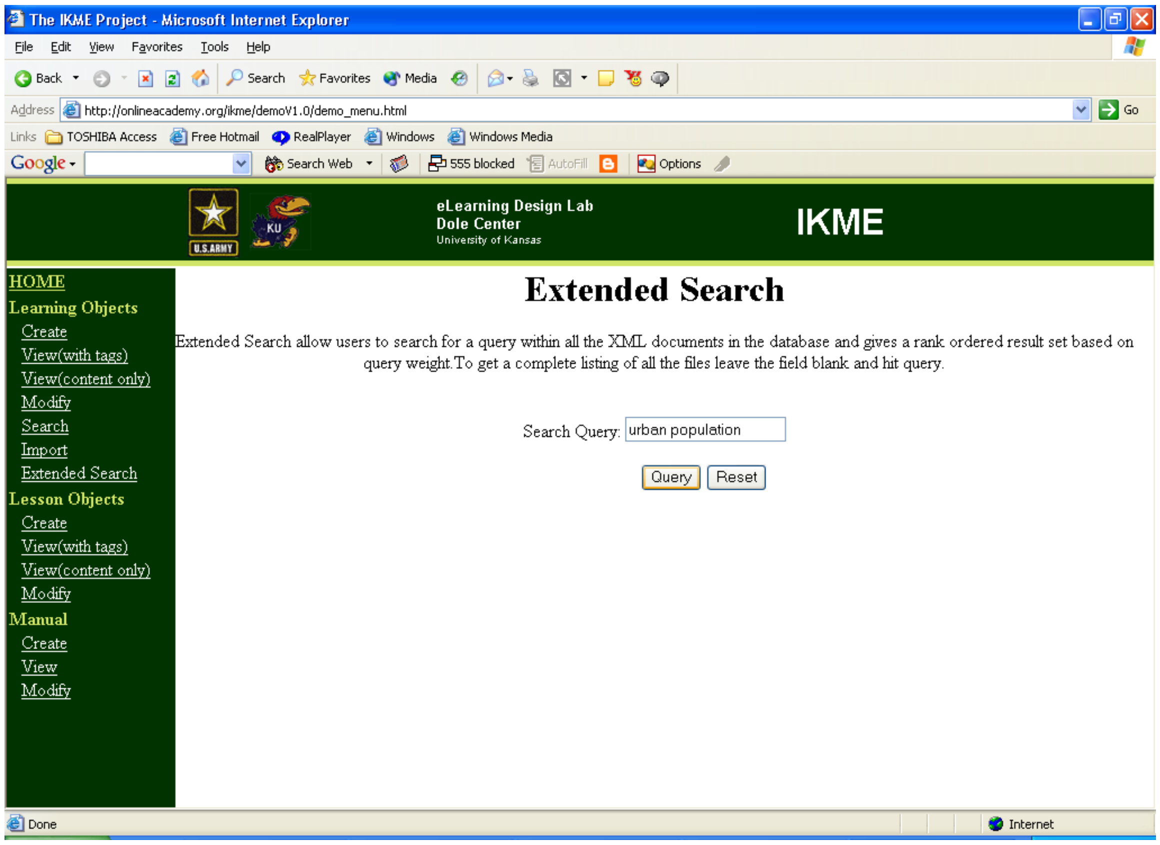
Manual
- Create
- View
- Modify

Number of hits : 17

No.	Document Name
1	An Urban Model
2	Characteristics of Urban Operations
3	Characteristics of Urban Operations
4	Civilian Concerns
5	General Population Size
6	Group Size, Location and Composition
7	Impact on Future Operations
8	Interests and Actions
9	Overview of the Urban Environment
10	Overview of the Urban Triad
11	Population Characteristics of Urban Areas
12	Residential Area
13	Residential Area
14	Segment Pattern
15	Transportation and Distribution Systems
16	Urban Operations
17	Urban Society

Figure 5.2 Result Set

There are 17 returned documents and all of them have both the terms.



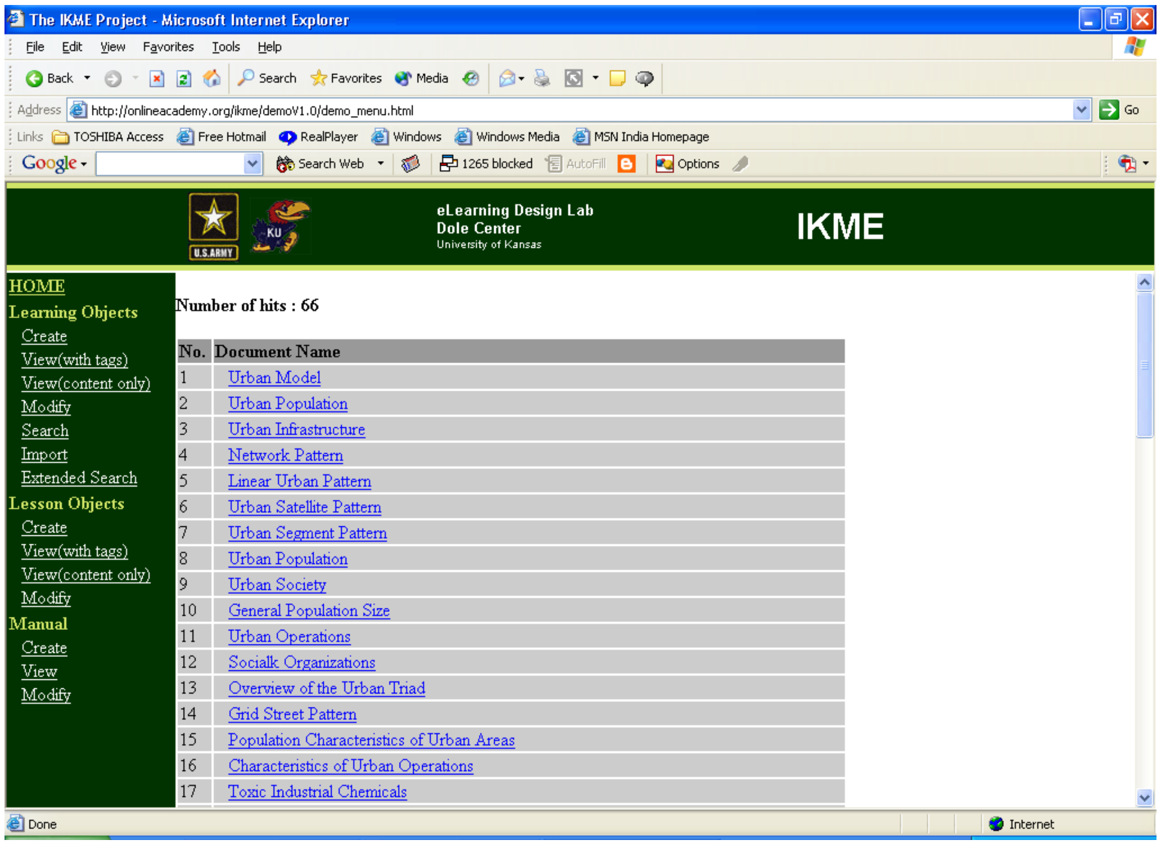
Extended Search

Extended Search allow users to search for a query within all the XML documents in the database and gives a rank ordered result set based on query weight.To get a complete listing of all the files leave the field blank and hit query.

Search Query: urban population

Figure 5.3 Extended Search with query

The extended search has also been given the same query as the previous search. Let us view the rank ordered result set.


The IKME Project - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://onlineacademy.org/ikme/demoV1.0/demo_menu.html
Go
eLearning Design Lab
Dole Center
University of Kansas
IKME
Done
Internet

HOME
Learning Objects
- Create
- View(with tags)
- View(content only)
- Modify
- Search
- Import
- Extended Search

Lesson Objects
- Create
- View(with tags)
- View(content only)
- Modify

Manual
- Create
- View
- Modify

Number of hits : 66

No.	Document Name
1	Urban Model
2	Urban Population
3	Urban Infrastructure
4	Network Pattern
5	Linear Urban Pattern
6	Urban Satellite Pattern
7	Urban Segment Pattern
8	Urban Population
9	Urban Society
10	General Population Size
11	Urban Operations
12	Socialk Organizations
13	Overview of the Urban Triad
14	Grid Street Pattern
15	Population Characteristics of Urban Areas
16	Characteristics of Urban Operations
17	Toxic Industrial Chemicals

Figure 5.4 Rank Ordered Result Set

Here we see that there are 66 documents returned. The 66 documents contain at least one query term or all of them. They are rank ordered and highly accurate.

- The entire implementation is done on the fly. All the processes undertaken to achieve the rank ordered result set is done after the user enters a query.
- The response time is very fast (less than one second). The project was tested for double the number of documents and still the response time was not any different (little over one second).
- The rank ordered result set obtained is extremely accurate, very robust and completely reliable.

6. Future Work

- The current project deals with 84 documents in the whole collection. This has also been tested for the response time with double the number of documents. Since the entire process is done on the fly, i.e. after the user enters the query, the response time could be longer when the number of documents becomes really large. This is an issue that should be taken care of.
- This rank ordered search feature is currently for the content tag alone within the XML documents. This could be extended for every other tag within the XML documents.

7. Conclusion

This project implements a complete content based searching with rank ordered retrieval in XML documents. The result set is perfectly rank ordered, accurate and extremely robust. As aimed the response time is extremely low and the entire process has been done on the fly. The primary goal and all the other related desired goals have been achieved.

Bibliography

[1] eXist Database Home Page

http://exist.sourceforge.net/

[2] XML:Twig Home Page

http://xmltwig.com/xmltwig/

[3] PERL programming language references

http://www.perl.com/

http://www.perldoc.com/

[4] XPath references

http://www.w3schools.com/xpath/default.asp

http://www.w3.org/TR/xpath20/

http://www.oreilly.com/catalog/xmlnut/chapter/ch09.html

[5] Learning Objects

http://www.eduworks.com/LOTT/tutorial/learningobjects.html
http://www.atl.ualberta.ca/downes/naweb/column000523.htm