Web Based Concept Hierarchy Viewer

Author

Pavan Kumar Ganjam

Committee

Dr. Susan Gauch, Chairperson

Dr. Jerry James, Committee Member

Dr. Perry Alexander, Committee Member

University of Kansas, Lawrence

Table of Contents

Abstract

The amount of web page content on the Internet has been increasing consistently. As a result, it is becoming increasingly difficult to find documents of interest to the user. Since then, there have been many attempts to improve search efficiency, some of which have been like categorizing documents against a predefined set of categories. KeyConcept [Madrid 2002] was one such project aimed at improving search by incorporating the concept matching into the search criteria. Documents are automatically classified to determine the concepts to which they belong. Query concepts are determined automatically either from their profile, from the short description of the query or by the user explicitly selecting them. Documents are finally retrieved based on both the keyword matching and the concept matching.

This project is aimed at building a dynamic navigation interface for the user profile of the KeyConcept project. The user profile is dynamically built by “watching over the user’s shoulder” as they browse the Web. The user profile, which is transmitted to the server, is a weighted ontology. This ontology is now displayed on the client browser as an expanding tree with the retrieved documents weighted and classified under the respective categories.

1. Introduction

1.1. Motivation

As the number of available Web pages grows, users experience increasing difficulty finding documents relevant to their interests. One of the underlying reasons for this is that most search engines find matches based on keywords, regardless of their meanings. To provide the user with more useful information, we need a system that includes information about the conceptual frame of the queries as well as its keywords. This is the goal of KeyConcept, a search engine that retrieves documents based on a combination of keyword and conceptual matching. Documents are automatically classified to determine the concepts to which they belong. Query concepts are explicitly entered by the user or automatically determined by means of a user profile; in this way, the system will be able to provide search results more relevant to the user's current activities and tasks.

1.2. Problems with the existing system

The existing version of KeyConcept uses a VC++ program to view the profile, limiting the viewing option to users of the Windows platform only. Since the user profile is an ontology or tree of concepts with associated weights, representing the profile as an expanding tree in the browser would make it available to users of all platforms. Concept hierarchies are also used by YAHOO, BIOT etc to present the information to the users

1.3. Goals and Contributions

- To develop a new user profile viewing strategy that replaces the existing VC++ version of the profile viewer that is limited to the Windows platform only.
- Make the profile creation generic so that it can be used by other applications also
- To change the existing profile directory structure to remove data redundancy.

1.4. Overview

The process of building the profile consists of 3 phases:

1) Training the classifier

2) Collecting user data and

3) Classifying the web pages from the collected URLs.

1.4.1. Training the Classifier

Training the classifier consists of indexing the training documents of each concept using the traditional *tf*idf* method. The result of indexing is 3 files: dictionary, postings and documents. Those files keep information about keyword frequencies for each concept and are used by the classifying agent to match the user documents with the closest categories. Essentially, the training phase creates an inverted index that stores, for each category, the centroid of the category.

1.4.2. Collecting the user data

In this phase, the URLs time when visited, and web page sizes are stored in a log file by a proxy server. The program extracts the URLs for each user, spiders the web sites that are considered to be relevant, and saves the web pages locally. A classifying agent processes all the html files collected by spidering the URLs the user has visited. The classifying agent is an expanded and improved Local Categorizing Agent (LCA) from the OBIWAN project [Zhu99]. It uses the dictionary, postings and documents files created in the training process. Each web page is treated as query and matched to the super document representing each concept. The classifying agent finds the top-matching super document and returns the corresponding concept ID from the ontology along with the weight of the match. For each concept in the ontology, its weight is calculated as sum of all its children's weights and its own weight. At the end of this process, the classifying agent returns the concepts with non- zero weights as the user profile.

1.4.3. Classifying the documents

All the html files collected by spidering the URLs the user has visited are processed by a classifying agent. The classifying agent is an expanded and improved Local Categorizing Agent (LCA) from the OBIWAN project [Zhu99]. It uses the dictionary, postings and documents files created in the training process. Each web page is treated as query and matched to the super document representing each concept.

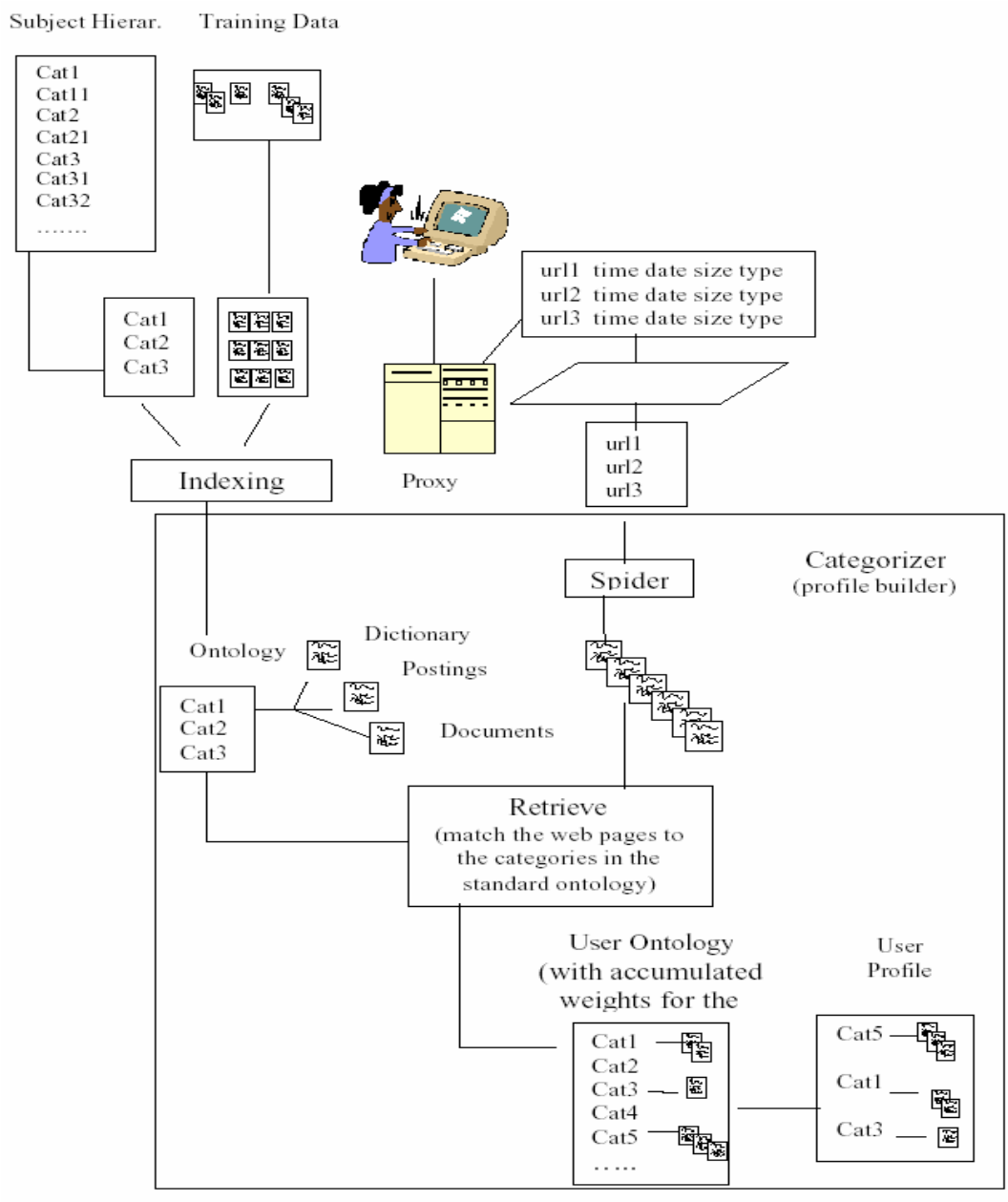


Figure 1. System Architecture

3. Design Issues

The screenshot of the Profile Viewer is shown below:

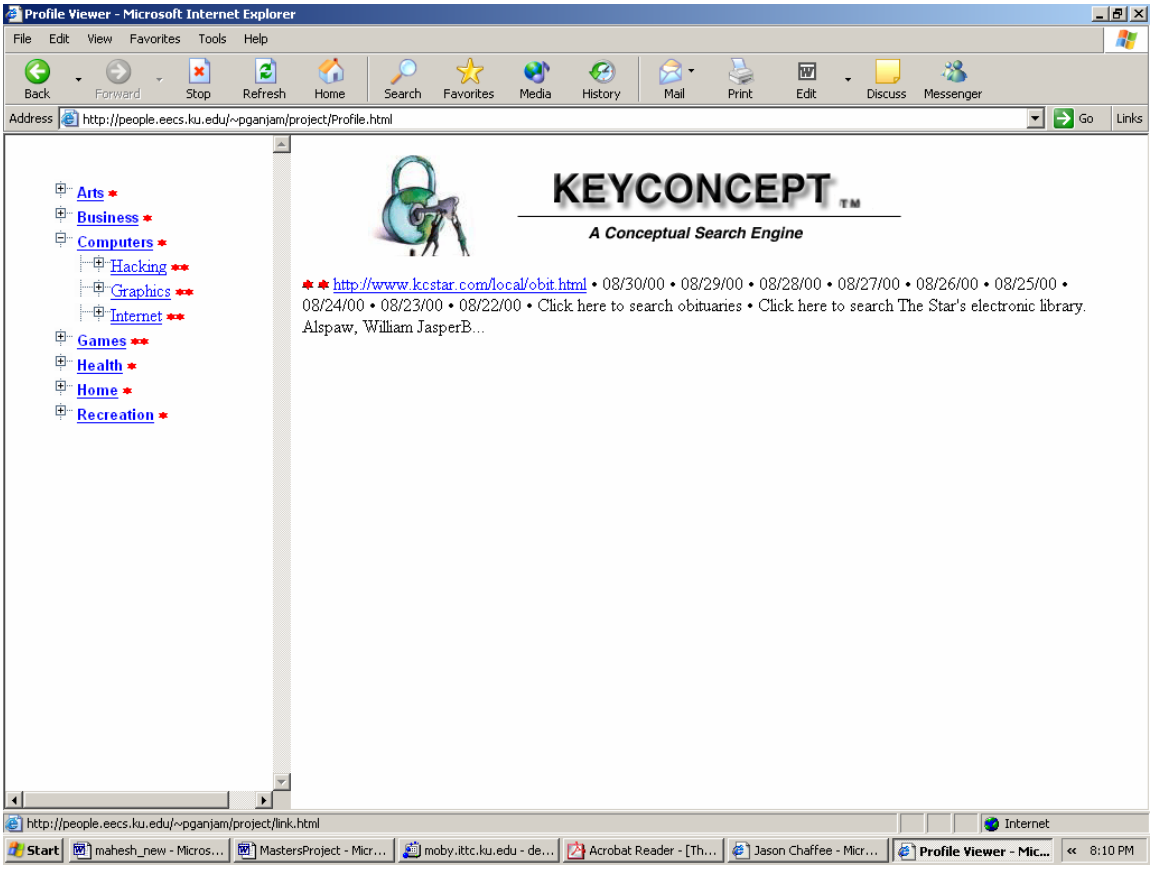


Figure 2. Profile Viewer

The objective of the *Profile Viewer* is to allow users to reach Web pages of interest to them. It uses the user profile represented as ontology to allow concept-based browsing of the indexed documents. The frame on the left of the screen contains the hierarchy of the user profile and the frame on the right of the screen will contain the actual Web pages and their summaries that were classified into that particular concept.

The user is able to click on concepts of interest and expand the tree to view lower levels until they reach a sub-concept that has Web pages classified into it. A relative weighting scheme is used to display the amount of content in each concept. The total weight of each concept is divided by the total weight of all sibling concepts. This relative weight is then used to assign anywhere from zero stars, little content compared to siblings, to five stars, a lot of content compared to siblings.

When the user profile becomes deep, it becomes inappropriate for display in a frame as shown in the screenshot. So, the viewer must be able to render the depth of the profile to n levels as specified by the user. The viewer must also be able to display parent concepts for back navigation when the profile is rendered to n levels.

4. Implementation Details

The system consists of two components.

1. Profile Generator
2. Profile Viewer

Profile Generator is a program on the client machine that monitors the user activities and generates the profile that is characteristic of the user. The user profile consists of a list of concepts and weights that represent the users interest in those concepts. The profile is now transmitted to the server where it is brought into the form of Standard Tree plus Weight.

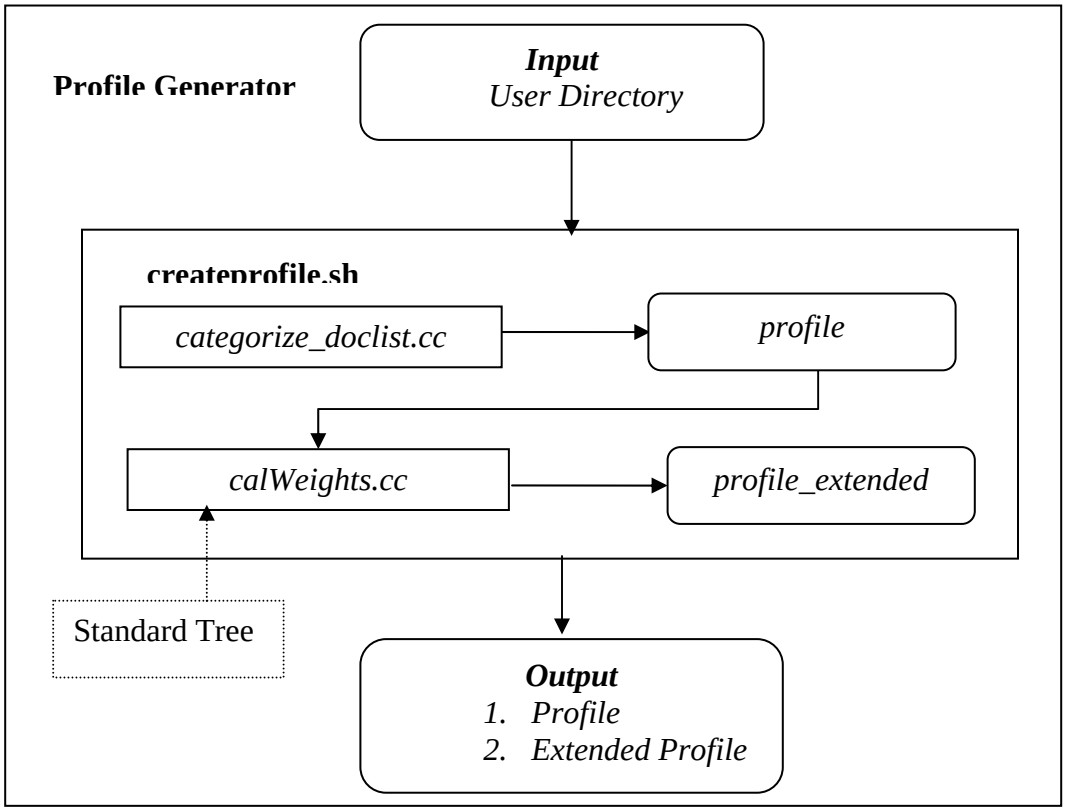


Figure 3. Profile Generator

Profile Viewer uses the user profile represented as an ontology to allow concept-based browsing of the indexed documents.

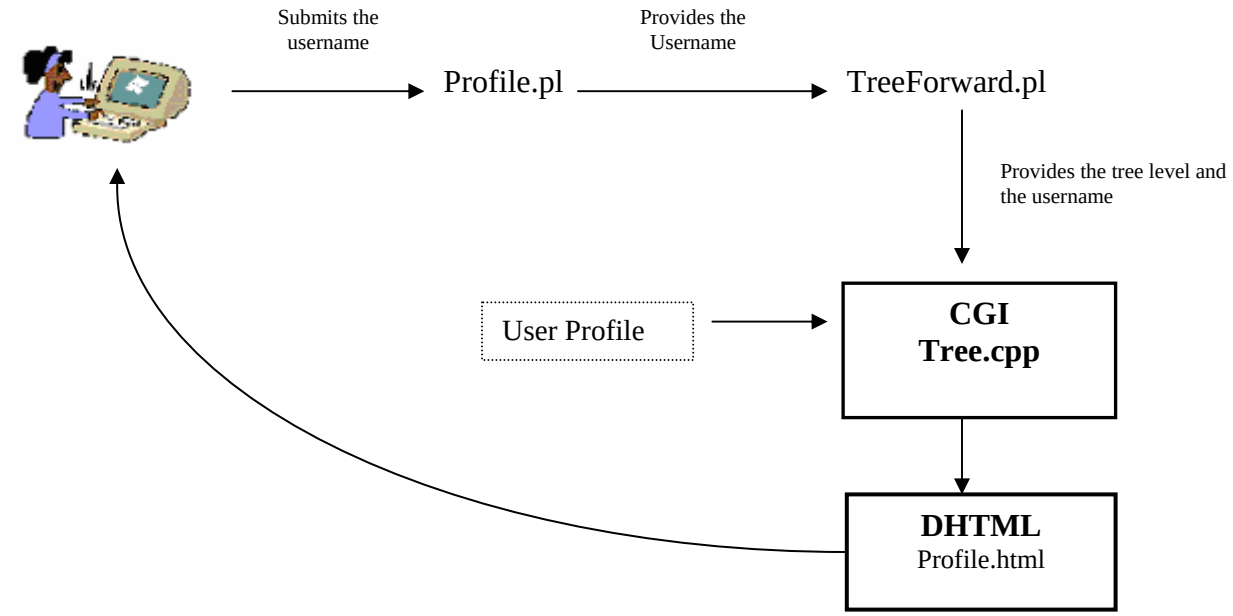


Figure 4. Profile Viewer

4.1. Profile Generation

4.1.1. CreateProfile.cc

Purpose: This program generates the user profile in the form of category plus weights.

Main program source file	createprofile/createprofilepgm.cc	
API Definition source file	createprofile/createprofile.cc	
Include file	include/createprofile.h	
API Function	void create_profile(prefilename, dict, post, docs, inputType, input, level, subjectTree, lcaHTMLsDir, output, maxURLs, numWords, numCat, prune_threshold, min_weightThold, update, date_processed)	
Input Parameters	dict	Path of the training dictionary file
	post	Path of the training postings file
	docs	Path of the training documents file
	prefilename	Path to file specifying document pre-processing options
	inputType	Specifies the format of the input 1 -- the input is a file containing a list of urls 2 -- the input is a file listing HTML files 3 -- the input is is a weight file
	input	Location of input (its type specified by inputType parameter)
	level	Level of the Standard tree
	subjectTree	Location of the Standard tree
	lcaHTMLsDir	Directory to store created file
	output	Path of the weighted standard tree to be created
	maxURLs	Maximum number of URLs

	prune_threshold	Minimum weight threshold for categories to be recorded in the output weighted std tree file
	min_weightThold	Minimum weight threshold (expressed as % to the weight of the most important category)
	update	1 - Update existing profile, 0 - Create new profile.
	date_processed	String specifying date when profile was created
Compilation instructions	In the root directory, type 'make profile'	
Execution instructions	Run the shell script bin/createprofile.sh	
Output file format	58635 3.0700 30155 2.0816 6380 1.0708 9609 0.1830 1033 0.1262 1038 0.1022 1036 0.0738 120812 0.0525	

Table 1. CreateProfile.cc

Comments: createprofile.cc uses urlFileNames, which is a list of document paths that the user has browsed. The urlFileNames file must be put in the user directory that also contains the urlFile.

Algorithm:

```
{
    If UPDATE flag is set
    {
        Update the existing profile
    }
    else
```

{

Create a new profile

}

Call categorizeDocList function

Call calWeights function

Create a new directory with the current date.

Move url and urlfilenames files into that directory.

Make a copy of the extended_profile in that directory.

}

4.1.2. Categorizedoclist.cc

Purpose: This program categorizes the input document into the top n categories of the standard tree.

Main program source file	createprofile/categorizedoclist.cc	
Include file	include/categorizedoclist.h	
API Function	void categorize_doclist (char *prefilename, char *tdict, char *tpost, char *tdocs, char *urlFilenames, char *urlFile, char *lcaHTMLsDir, int inputType, char *results, int numTopWords, int numCat, int threshold, long &count, float &avgWt, int update)	
Input Parameters	prefilename	Path to file specifying document pre-processing options
	tdict	Path of the training dictionary file
	tpost	Path of the training postings file
	tdocs	Path of the training documents file
	urlFilenames	Path of the file that has the listing of all documents that the user visited.
	urlFile	Path of the file that lists the urls visited

	inputType	Specifies the format of the input 1 -- the input is a file containing a list of urls 2 -- the input is a file listing HTML files 3 -- the input is a weight file
	lcaHTMLsDir	Directory to store created file
	results	Path of the file that stores the categorizer results.
	numTopWords	Total number of words used to categorize the documents
	numCat	Number of levels of the standard tree to be considered when categorizing the document.
	threshold	Threshold to be used when categorizing a document
	count	The count of categories into which the document was categorized
	avgWeight	Average weight of all category weights
	Update	Update Flag
Execution instructions	Called by createprofile	
Output file format	58635 3.0700 30155 2.0816 6380 1.0708 9609 0.1830 1033 0.1262 1038 0.1022 1036 0.0738 120812 0.0525	

Table 2. Categorizedoclist.cc

4.1.3. CalWeights.cc

Purpose: This program modifies the profile into the form of standard tree plus weight that is suitable for display.

API Definition source file	createprofile/calWeights.cc
Include file	include/calWeights.h
API Function	void calWeights(stdTree, weightsFile, results)

Parameters	stdTree	Location of the Standard tree
	weightsFile	Path of the raw profile file generated.
	results	Path of the extended profile to be created for display
Output file format	Top/Home/Cooking/Baking_and_Confections 006005052000000000 58635 100 Top/Home/Cooking 006005000000000000 201 100 Top/Home 006000000000000000 7 45.4135 Top 000000000000000000 1 100 Top/Sports/Basketball/Professional 014011002000000000 30155 100 Top/Sports/Basketball 014011000000000000 404 100 Top/Sports 014000000000000000 15 30.7924 Top/Computers/Programming/Compilers 003016006000000000 6380 100 Top/Computers/Programming 003016000000000000 126 100 Top/Computers 003000000000000000 4 15.84 Top/Reference/Education/Colleges_and_Universities 009001012000000000 9609 100 Top/Reference/Education 009001000000000000 243 100 Top/Reference 009000000000000000 10 2.70706 Top/Arts/Music/Styles 001001001000000000 1033 41.7604 Top/Arts/Music 001001000000000000 27 100 Top/Arts 001000000000000000 2 4.47035 Top/Arts/Music/Instruments 001001006000000000 1038 33.8187 Top/Arts/Music/Collecting 001001004000000000 1036 24.4209 Top/Shopping/Tools/Gardening 012019002000000000 120812 100 Top/Shopping/Tools 012019000000000000 96725 100 Top/Shopping 012000000000000000 13 0.776616	

Table 3. CalWeights.cc

Algorithm:

{

Store Standard Tree as a Hash table indexed on the location field.

Store the location fields in a lookup array indexed on the Category ID's.

Open the profile file

For each row entry in the input file

{

Add the weight value to the existing value in the hash table for the corresponding category and its parent categories.

}

Close the profile file.

Update the weight values with the percentage per-sibling weight or the total weight based on the WEIGHT_FLAG.

Write the extended profile file to the output directory

Sort the extended profile file based on the location field.

}

4.2. Profile Viewer

4.2.1 tree.cpp

Purpose: Generates the DHTML for display in the browser.

Algorithm:

{

Initialize *item_clicked* with the value read from the standard Input

If the value read is “000”

{

MIN_SIZE = 3

MAX_SIZE = nLevels * 3

}

Else

{

MAX_SIZE = item_clicked.size() + 3;

MIN_SIZE = MAX_SIZE - (nLevels-1)*3;

}

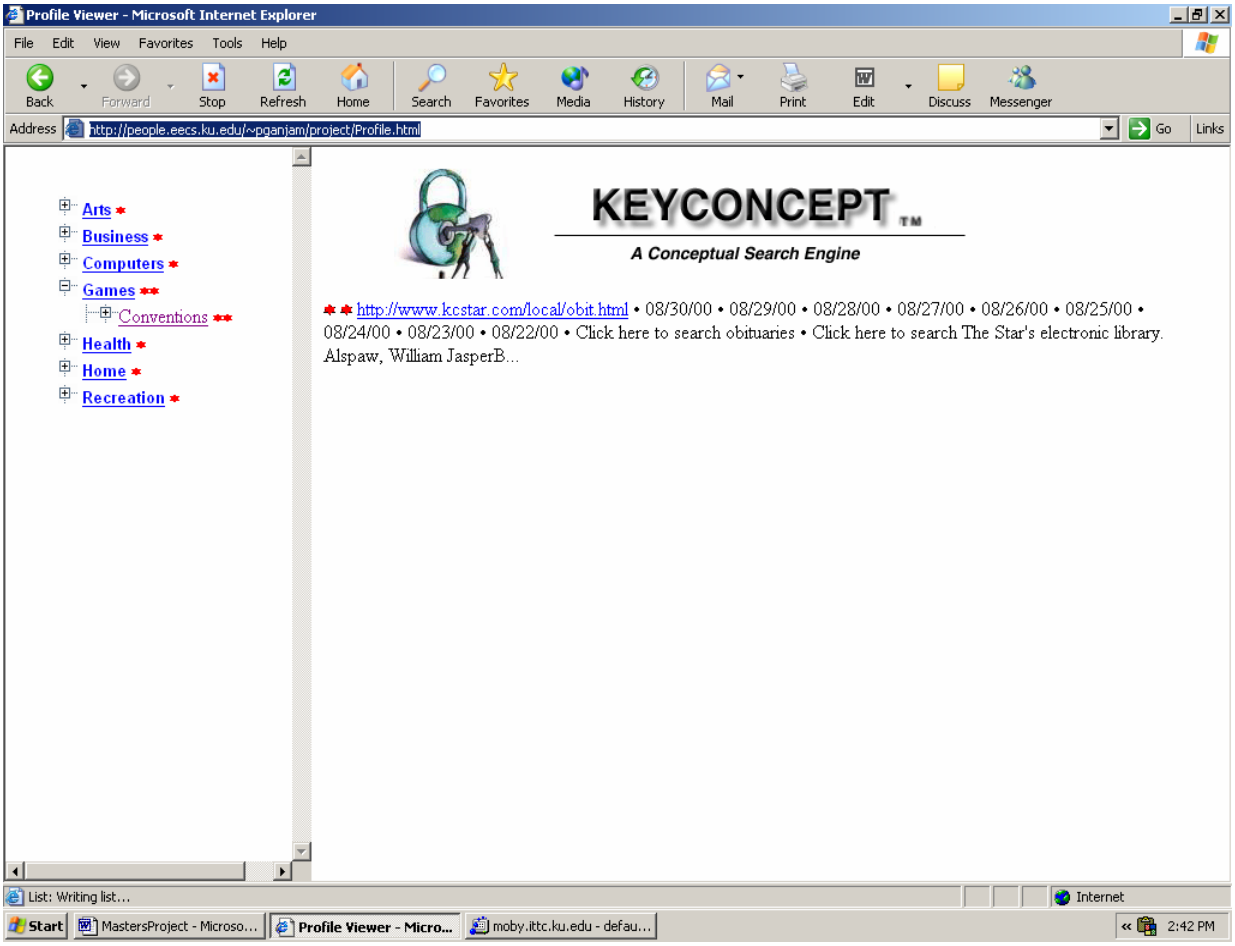
```
        Read current node from the file
        While (Read next node from the file)
        {
                If current node location prefix > MAX_SIZE
                        Continue;
                If next node is child of current node
                {
                        Case LEVEL #1 node:
                                If current nodes prefix size == MAX_SIZE
                                        Add Current Node to the ROOT Node as leaf
                                Else
                                        Add Current Node to the ROOT Node as list
                        Case NON-LEAF other node:
                                 If current nodes prefix size == MAX_SIZE
                                        Add Current Node to its parent node as leaf
                                Else
                                        Add Current Node to its parent node as list
                        Case LEAF NODE
                                        Add Current Node to its parent node as leaf
                }
        }
        Add Current Node to its parent node as leaf
}
```

4.2.2. list.js

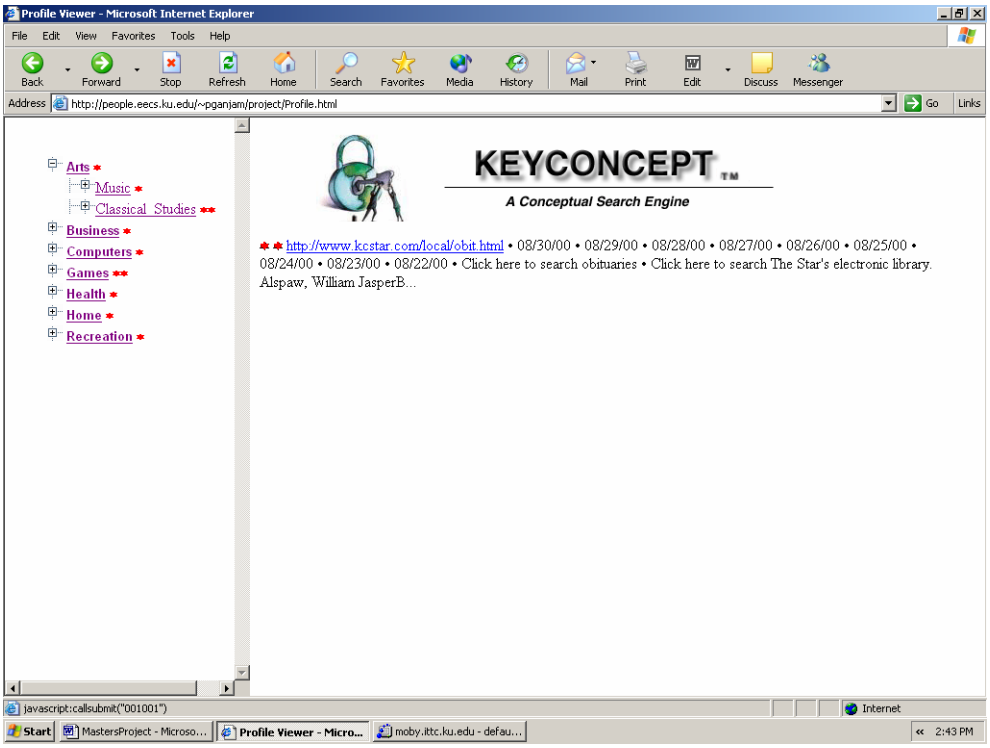
Script File	profileviewer/list.js
Purpose	The JavaScript extension contains functions that are responsible for the expansion of the tree as the user navigates through the profile. It also contains calls to refresh the page for nodes on the tree that go beyond the specified depth.

Table 4. List.js

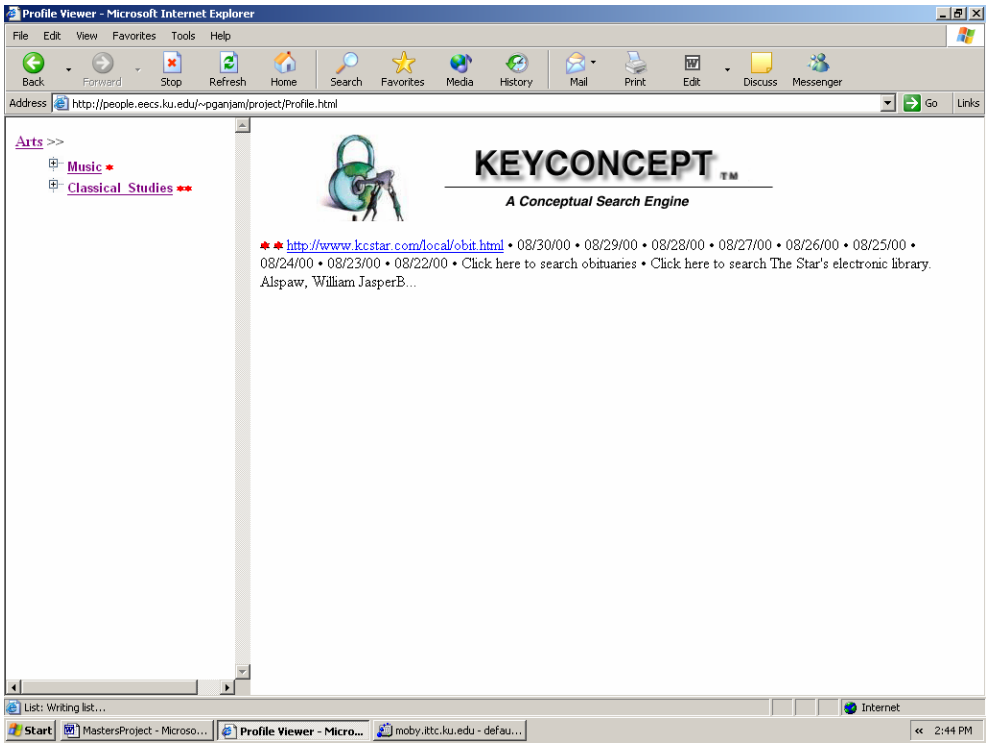
5. Screenshots

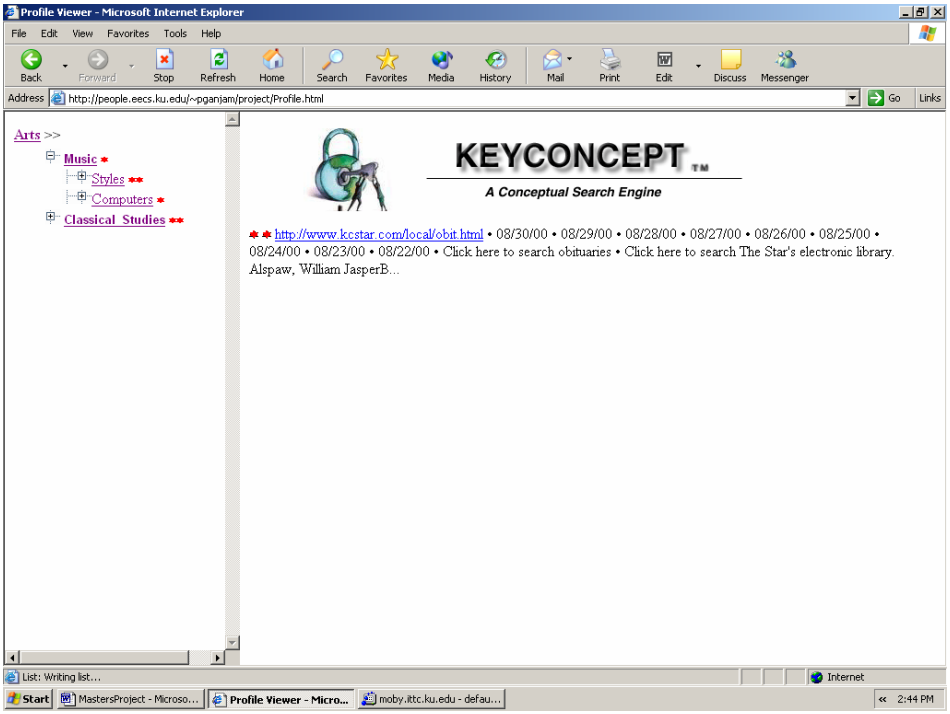


Comments: The user profile is represented as an ontology to allow concept-based browsing of the indexed documents. The user can browse the ontology to reach topics of his interest.

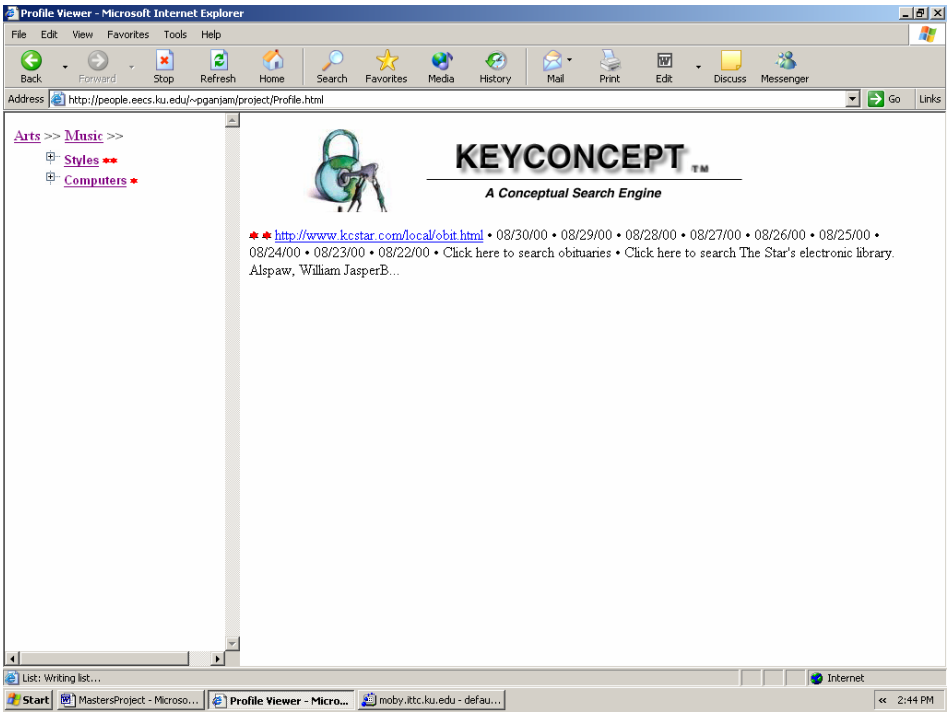


Comments: The depth of the profile viewer being set to 2, the system must render the viewer to two levels only





Comments: When the depth of the tree goes beyond the specified level, the tree is rendered and the parent concepts are displayed on the top of the page for back navigation.



6. Conclusions and Future Work

The primary goal of building a web based concept hierarchy has been achieved. A generic and smaller profile has been implemented. The project can be used as a tool for viewing ontologies. Representing the user as an expanding tree is in keeping with the ontology structure of the user profile. Different methods to transfer the profile to the server were studied and the profile was decided to be sent via ftp.

Better approaches to find similarity based on ontology matching can be explored. The ontology viewer currently uses the profile format used by the Keyconcept project. Ontologies are popularly represented using the OWL language. So to be able to view ontologies written in the OWL language a plugin may be developed.

7. References

1. **“Ontology-Based User Profiles for Search and Browsing”,** Susan Gauch, Jason Chaffee, Alexander Pretschner. Submitted to User Modeling and User-Adapted Interaction (UMUAI) journal, June 2002.

2 **"User Profiles for Browsing Biodiversity Information"**, Susan Gauch, BDEI PI meeting, Washington, D.C., Feb. 10 - 11, 2003.

3 **“Personalization on the Web”**, Alexander Pretschner, Susan Gauch, 1999

4 **“Ontology Based Personalized Search**”, Alexander Pretschner, MSc. Thesis. Kansas University 1999

Appendix A

Directory Structure

```
KeyConcept-v2.1
|…….. CreateProfile
|         |……..CreateProfile.cc
|         |……..CalWeights.cc
|         |……..ScanWeights.cc
|
|…….. ProfileTestFiles
|         |……..Users
|         |       |……..User1
|         |       |……..User2
|         |       |……..User3
|         |……..Files
|                 |…….. Dict
|                 |…….. Post
|                 |…….. Docs
|                 |…….. StdTree
|
|…….. ProfileViewer
|         |……..Tree.cc
|         |……..Profile.pl
|         |……..TreeForward.pl
|
|…….. Webcode
|         |……..tree.html
|         |……..list.js
|
|…….. Binary
|         |…….. Binaries and Shell Scripts
|
|…….. MakeFile
```