TECHNICAL UNIVERSITY OF CRETE
DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

Multimodal Interfaces for Web Information Retrieval

BY
NIKOLAOS PALLAS

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DIPLOMA IN
ELECTRONICS AND COMPUTER ENGINEERING
AT
TECHNICAL UNIVERSITY OF CRETE
CHANIA, GREECE
OCTOBER 2005

TECHNICAL UNIVERSITY OF CRETE

DEPARTMENT OF
ELECTRONICS AND COMPUTER ENGINEERING

The undersigned herby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled “Multimodal interfaces for web information retrieval” by Nikolaos Pallas in partial fulfillment of the requirements for the degree of Diploma in Electronics and Computer Engineering.

Supervisor:

____
Prof. Alexandros Potamianos

Board Members:

____
Prof. Vasilis Digalakis

____
Prof. Euripides Petrakis

TECHNICAL UNIVERSITY OF CRETE

Date: October 2005

Author:	Nikolaos Pallas
Title:	Multimodal interfaces for web information retrieval
Department:	Electronics and Computer Engineering
Degree:	Diploma
Submitted:	October 2005

____

Signature of Author

Table of Contents

Table of Figures

Abstract

The use of devices such as mobile phones and PDAs, for applications formerly hosted only on a classic computer, has introduced the issue of multimodality. A multimodal application is an application that combines different types of input/output (mouse/keyboard action, text, voice, etc.). Search engines have proven to be the most important tool for users to navigate and exploit the content available in the World Wide Web. In this thesis we implemented a multimodal search engine.

Our search engine has features of 3rd generation search engines. Those features are clustering and summarization support. Clustering is the categorization of the search results in groups. Summarization is the creation of a summary for a result or a set of results.

In general, this thesis deals with the creation of a search engine, the implementation of the clustering and summarization mechanisms and the design of a functional user interface that supports multimodal input/output. The algorithms we used are analyzed, the design constraints demonstrated and the choices we made explained.

Acknowledgements

I would like to thank Alexandros Potamianos, my supervisor for his guidance and patience during the elaboration of this thesis. Special thanks, to all post-graduate students of the Telecommunications Laboratory of the Technical University of Crete for providing support and help relevant to this work. I also would like to thank everyone, troubled or not regarding this thesis.

Great thanks to all friends I've made in Chania; George, Fanouris, Stavros, John, Dimitris (Mitsos), Dimitris (Kyb), Akis and Chris. You made things a whole lot easier…

Above all I want to thank my family; my father Kostas, my mother Froso, my brother George, my sister Katerina and my aunt Antonia for their love, support and patience (double thanks for that).

Finally, thanks to anyone who, at some point, thought of waking me up before 11 a.m. but never did.

Chapter 1

INTRODUCTION

1.1 Intro

During the past few years, more and more devices support applications, formerly suitable only for home computers. The nature of these devices, being usually mobile, and used for other reasons as well, has introduced the issue of multimodality for their applications. Multimodality is about not narrowing input/output capabilities to a specific form, but being able to use different forms of it in parallel. Thus, multimodality offers easier and more handy ways of human-computer interaction. Some of the most common types of input are: mouse/keyboard action, voice, text, touch-screen; while for output, graphic display, text, speech and video are the most common. A combination of two or more of the above is the foundation of a multimodal application.

Our goal in this thesis is to develop a multimodal application and specifically a multimodal search engine. A search engine is probably the most important tool a user has, in order to browse the endless amount of information available in the web, and it would be useful for him to handle it in different ways, using alternative devices. Along with the features of multimodality, our purpose is to embody features of 3rd generation search engines so that our application can be considered relatively modern. The features we use are those of clustering and summarization of results. Clustering is about grouping results into clusters, while summarization is extracting summaries out of them. As we will see in this thesis, those features not only offer more information to the user but help us both to build a better user interface and to overcome specific problems that arise.

In general, this document contains: all the issues we had to take in mind for creating the application, the design procedure, the algorithms we implemented, the tools we used in our implementation and an analysis of the user interface and how it works.

Later in this chapter we present a first approach to creating the application focusing on the implementation we shall follow, the arising problems and how we are going to solve them, and the related work on the fields of interest. Moreover we supply some use cases of how the desired application should be.

Chapters 2, 3 and 4 are about the back-end of the application. In particular, chapter 2 presents all steps we had to take in order to get results after a user query; that includes creating a database of web pages, storing their attributes in an index and using search and ranking algorithms to retrieve the most relevant results of a query. Chapter 3 is about the clustering mechanism and the creation of a structure that acts as a link between the back-end and the front-end. The procedure of creating the clusters is presented and after that, the implementation of the structure is explained. Finally chapter 4 shows what algorithms we use to extract summaries from single web pages or groups of web pages.

Chapters 5 and 6 offer an analysis of the front-end. In Chapter five we discuss matters concerning the specification of the multimodal user interface, the whole application architecture and the design concessions we made. Chapter 6 shows the implementation of the final application, and usage examples concerning it.

Finally Chapter 7 presents future work based on our designed application and the field of research in general. Appendix A shows a sample use of the application, while Appendix B, acting more like a guide, shows the steps we took in order to set up the application.

In general, the work done during the elaboration of this thesis is briefly described below:

- We analyzed all aspects of creating a multimodal application and took the necessary decisions.
- We created a repository of web pages and indexed some of their attributes.
- We used searching and ranking algorithms (contributed by Epimenides Voutsakis) in order to retrieve query relevant web pages from the repository.
- We used *spkmeans* program (slightly changing its source code) in order to create cluster and assign the documents into them.
- We created the structures withholding information regarding cluster formation and result data.
- We implemented algorithms for extracting summary out of a web page and a description out of a cluster of web pages.
- We used MEAD summarization toolkit to extract summary out of a cluster of web pages.
- We developed a web user interface with multimodal support features.
- We created the links between the user interface and the back-end.
- We tested the user interface for different usage scenarios.

1.2 First Approach

Before starting to implement our application, it is vital that we first analyze it and make all necessary decisions regarding the desired result and the arising issues. In general, the use of a multimodal search engine can be described by the following paragraph: “The user sends a query to the system, which retrieves all relevant results and presents them back to the user in a browsable way. The user can navigate through results and retrieve additional information about them. The communication between the user and the system is done in varying ways”.

In order for all this to happen, a lot of processing is required. The biggest part of this processing will take place at the back-end. It will be responsible for retrieving search results, organizing them in a structure and handling additional search engine functionality available by the front-end. As for the front-end, its job will be to offer the multimodal user interface and coordinate the operation of the back-end.

Regarding the target host devices for our application, we focus on mobile devices like mobile phones and palms. This comes as a result of the continuously increasing use of such devices; this makes our application even more efficient. Our application having multimodal support means that the implementation should reflect the handling of different types of input/output. In particular we will follow a design according to the requirements of both telephone and palm devices, lighting up each case whenever there is a diversification between them.

Figure 1.1 shows a general idea of the architecture that will be followed to create the application. The user with his device interacts with the system, which sends the query to a search engine. The search engine retrieves relevant results and sends them to a clustering engine to distribute them into groups. Then they are forwarded back to the system. The system runs some more processing and the final result is presented to the user.

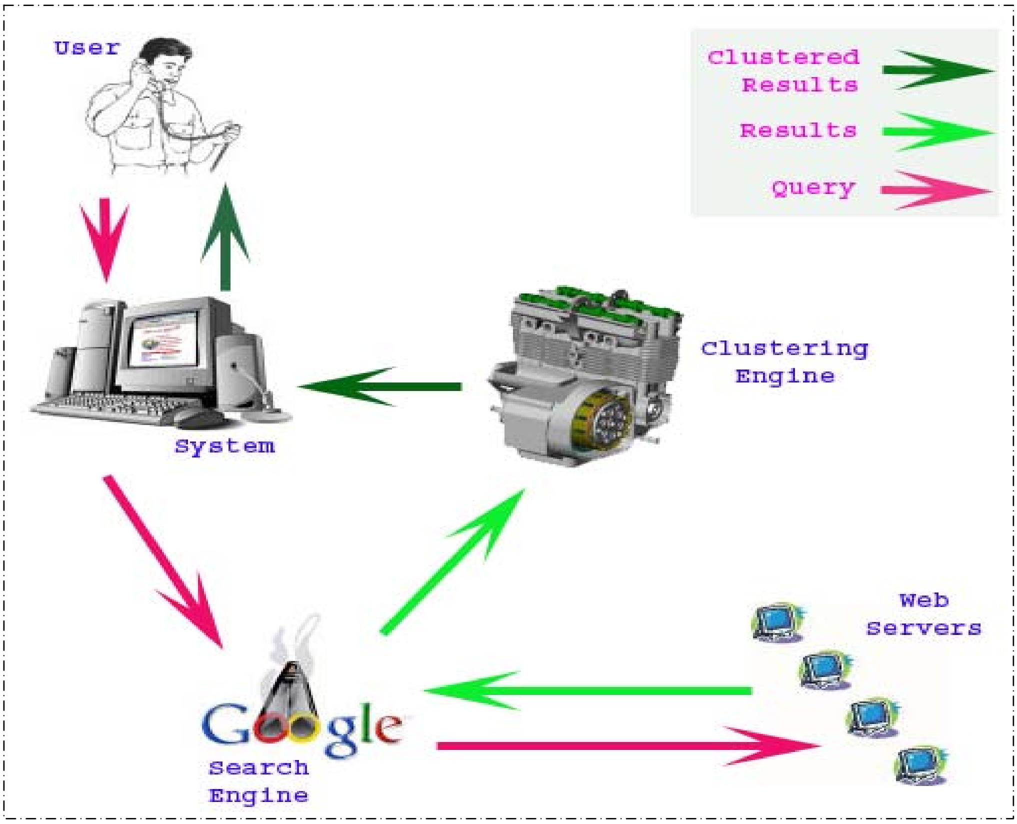


Figure 1.1: Application Architecture in General

1.2.1 Dealing with Constraints

The most important thing to determine our design is to deal with all critical issues arising when building such an application. The bandwidth and space limitations, the necessity of quick response to the user, the quality of provided functionalities and the way the results should be displayed designate what we have to face:

1. **Bandwidth:** The main problem for a mobile device is the bandwidth available by its network. Contrary to home computers, the bandwidth is severely limited. This makes extremely necessary to provide results in such way, that information reduced in size but not in content is transmitted.

2. **Speed of response:** This is usually required in every case, but in our case it is vital. Imagine someone waiting a long time over the phone for the query results. Or someone using his palm's/mobile's network access, waiting for response under high charging rates. Probably he would stop the interaction after a while cursing the money it cost. Thus, we have to minimize the time concerning the retrieval and display of the results. This can be done by reducing pre-processing time, which means we need quick algorithms in our back-end.
3. **Quality of results:** This is needed so that we can produce a decent reply to the user, driven by what he asked for. It is a measure of quality for the whole application, therefore the better the results, the better our application is. It is achieved by using efficient algorithms for searching and clustering and good summarizers for the result and cluster summaries.
4. **Space Limitations:** This has to do with the presentation of the results. A speech output device (e.g. phone) has to use short prompts and replies. A display device (palm, mobile phone) has a limited size screen, which again means we need short prompts and replies. Moreover it means that we have to output the results in parts, not the whole list all at once.
5. **Simple User Interface:** In combination with the previous issue, it is obvious that a device specific user interface has to be designed. It has to be simple for the user to use and it should emphasize on available functionality and the synergies between modalities.

The solution of the above matters is a gnomon for our future implementation and thereby we have them constantly in mind.

1.2.2 Our Approach

Now, let’s take a deeper look into the application we are going to build, explain what the features of clustering and summarization offer and how they are used. This will be done by explaining the basic procedure of searching using the system.

The User Interface asks the user for the search query, and once it gets it, it initiates a search, which takes place in the back-end: The query is submitted to the search engine which retrieves from the available web page repository all relevant results, ranked in terms of relevancy. Instead of instantly presenting the results to the user via the UI, the clustering engine barges in. Its purpose is to create clusters from the results and distribute them into those clusters. In fact, that is done iteratively (hierarchical clustering) so that we get a tree structure of clusters and results. We should note that those clusters don’t pre-exist but are created based on the content of the result set.

Now the UI presents clusters instead of results to the user. In order to get to the results, the user has to navigate through cluster hierarchy. But to do so, a certain naming for the clusters has to be done; that is done by the summarization module. The summarization module is responsible for extracting various summaries, and cluster description is one of them. Cluster description is actually a few-word phrase descriptive to the clusters contents. Additional to this, a cluster summary is created too, should the user ask for it. This summary is a few sentences long, again having to do with the clusters contents. Navigating through the hierarchy, the user finally reaches the selected cluster’s results. For every result, the summarization module provides a summary too. As it is obvious, clustering and summarization play an integral part in our application. The following figures (1.2 and 1.3) present in what way clustering and summarization respectively benefit both our implementation and the user interaction. (No need to include the effect of application benefits into user benefits, since it’s obvious that their effect is of high importance).

CLUSTERING	
Application benefits	**User benefits**
1) Separates the results and their display in parts, which reduces bandwidth needs 2) Clusters need less space and time to display than results 3) Helps us create a cluster based User interface, which is rather simple and effective.	1) User doesn't have to browse through a long list of results. They just have to follow the easy to navigate cluster hierarchy. 2) Instead of browsing a wide in topic list of results, users can select the cluster that is more nearer to what they really search for, and get only its results.

Figure 1.2: Clustering Benefits

SUMMARIZATION	
Application benefits	**User benefits**
1) A summary in general reduces the corresponding information, therefore it reduces space and bandwidth needs. 2) It helps us build a simpler and more functional user interface	1) User can get a glimpse of a cluster's topic by its description. 2) By listening to a cluster's summary, the user knows what the cluster's results are about; this can save him time. 3) A webpage's summary provides him with the most important content of the webpage, so he doesn't need to read/listen to the whole of it.

Figure 1.3: Summarization Benefits

Bearing in mind the limitations previously discussed, another issue arises. It has to do with the structure of clusters we create. As we already mentioned, our target is to create a tree structure, but we have to define how deep this structure should be and how much it should expand. This must derive from the expected time it takes to display a cluster or a result. We estimate that a cluster having up to 5 sub-clusters or results is good enough to present. Moreover the results shouldn't be deep below the initial clusters; a hierarchy of 3 levels of clusters is more than enough. So, our tree structure should have 3 levels of clusters with branching factor 5.

Regarding the user interface, we need just a limited functionality interface (the only thing the user can do is search after a query). A support system offering context-sensitive help to the user is however a necessary addition. Moreover, it has to support multimodal input/output and to communicate with the mechanisms we've just discussed. The next topic presents sample use cases for an application similar to what we're trying to implement.

1.2.3 Use Cases

The following use cases show the example use of a system offering search functionality with clustering and summarization features.

- The first use case we present is about a voice interface for our application. During this scenario, the user enters a query and navigates through the results. It also shows the help system in action. It is presented for speech input/output.

(S) Welcome to the voice search engine. For any help during the procedure say Help.
(S) Please enter the search string for your query.
(U) “Maradona”
(S) There are 12123 results for this search string. Please choose desired category for your query about “Maradona”. Available categories are: 1) Diego Armando Maradona, 2) Argentina, 3) Football, 4) Greatest Players, 5) Fidel Castro.
(U) Help!
(S) In order to choose a category, say the name of the category or press the number of the category.
(U) “Football”
(S) You have selected “Football”. Please choose desired subcategory. Available subcategories are: 1) Naples FC, 2) Boca Juniors FC, 3) Campionato, 4) World Cup. Say ‘Back’ to return to the previous level.
(U) “World Cup”
(S) You have selected “World Cup”. Please choose desired subcategory. Available subcategories are: 1) World Cup Stars, 2) Mexico 1986, 3) Buenos Aires 1978, 4) Hand of God. Say ‘Back’ to return to the previous level.
(U)Back.
(S) You have selected “Football”. Please choose desired subcategory. Available subcategories are: 1) Naples FC, 2) Boca Juniors FC, 3) Campionato, 4) World Cup. Say ‘Back’ to return to the previous level.
(U) “Naples FC”
(S) There are 3 results for this subcategory. To browse between results say next or previous. To exit, say Exit.
(S) 1) Naples FC Official site. ‘www.napolefc.com’
(U) Next.
(S) 2) Naples FC fans site. ‘www.napolefcfans.com.’
(U) Next.
(S) 3) Naples FC greatest players. ‘www.napolefcstars.com’.
(U) Previous.
(S) 2) Naples FC fans site. ‘www.napolefcfans.com.’
(U) Exit.
(S) Would you like to start a new search?
(U) No.
(S) Thank you for using the voice search engine.

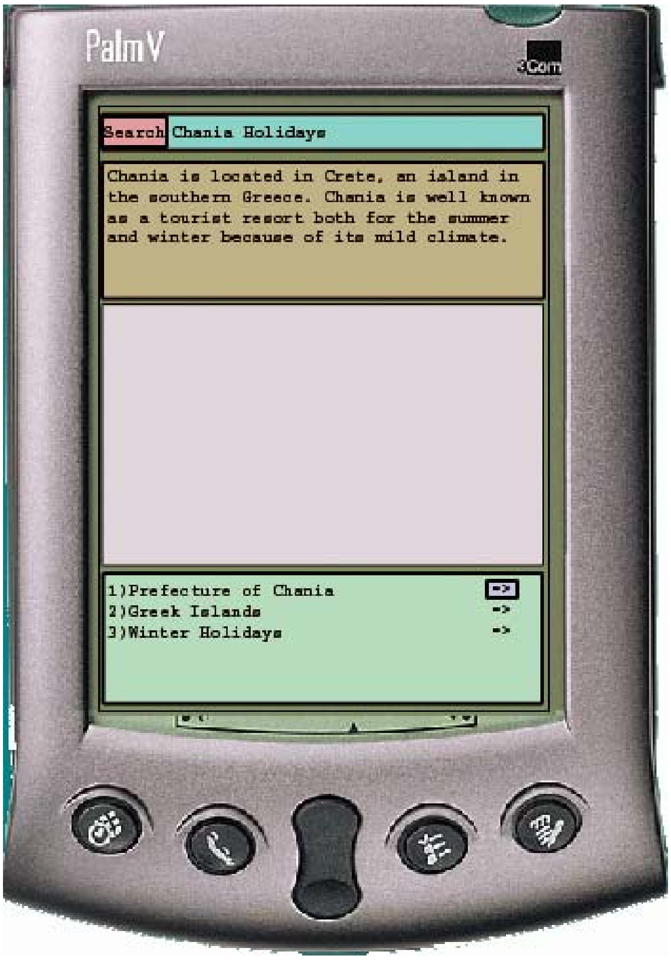
Figure 1.4: Example Use Case 1

- The second use case is similar to the previous one, but it emphasizes on the summarization feature. It still is speech I/O specific.

(S) Welcome to the voice search engine. For any help during the procedure say Help.
(S) Please enter the search string for your query.
(U) “Chania Holidays”
(S) Chania is located in Crete, an island in the southern Greece. Chania is well known as a tourist resort both for the summer and winter because of its mild climate. Select appropriate category to find out more about 1) Prefecture of Chania, 2) Greek Islands, 3) Winter Holidays.
(U) “Chania”
(S) The capital of the prefecture of Chania is the city of Chania. For further information you could visit the official website of the municipality of Chania at www.chania.gr. In summer many waterfronts like Platanias, Paleochora and Sfakia attract thousands of tourists every year. Available subcategories to narrow your search are 1) City of Chania, 2) Travel Guides of Chania, 3) Platanias, 4) Paleochora, 5) Sfakia
(U) “Platanias”
(S) Platanias lies 16 km to the west of the city of Chania. By its long sandy beach stand big hotel units, restaurants, beach bars and night clubs. For more information visit www.platanias.gr or select one of the following subcategories: 1) Accessibility, 2) Hotels in Platanias, 3) Restaurants in Platanias, 4) Nightlife in Platanias.
(U)”Hotels”
(S) There are 3 websites that matched your query. To browse between results say next or previous. To exit, say Exit. Ask for “More Info” in order to get more information on a specific result.
(S) 1) Apladas Hotel. A first rate hotel complex, offering a special experience for your summer holidays. ‘www.apladas.gr’
(U) “More Info”.
(S) Brand new and imposing APLADAS BEACH is located directly on the sandy beach of Platanias village in Chania-Crete. APLADAS BEACH with its' simple and chic design, high quality services, the relaxing atmosphere and the high standard facilities, focuses on making summer holidays an unforgettable experience.
(U) Exit.
(S) Would you like to start a new search?
(U) No.
(S) Thank you for using the voice search engine.

Figure 1.5: Example Use Case 2

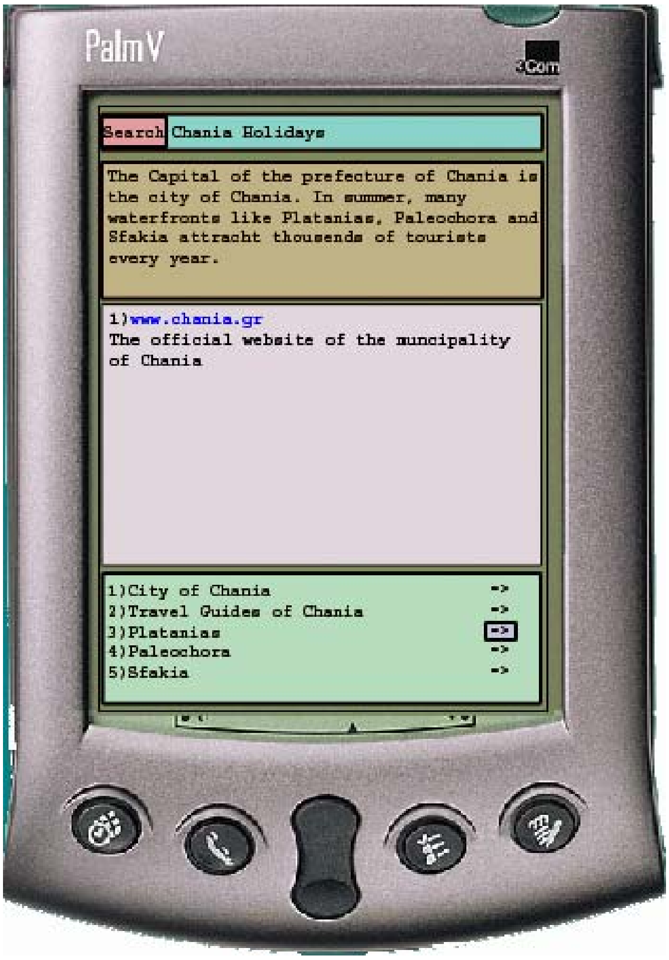
- The final use case (fig 1.6) concerns a similar interface as the previous one, but designed for using a palm this time.



Search Chania Holidays

Chania is located in Crete, an island in the southern Greece. Chania is well known as a tourist resort both for the summer and winter because of its mild climate.

1)Prefecture of Chania ->
2)Greek Islands ->
3)Winter Holidays ->

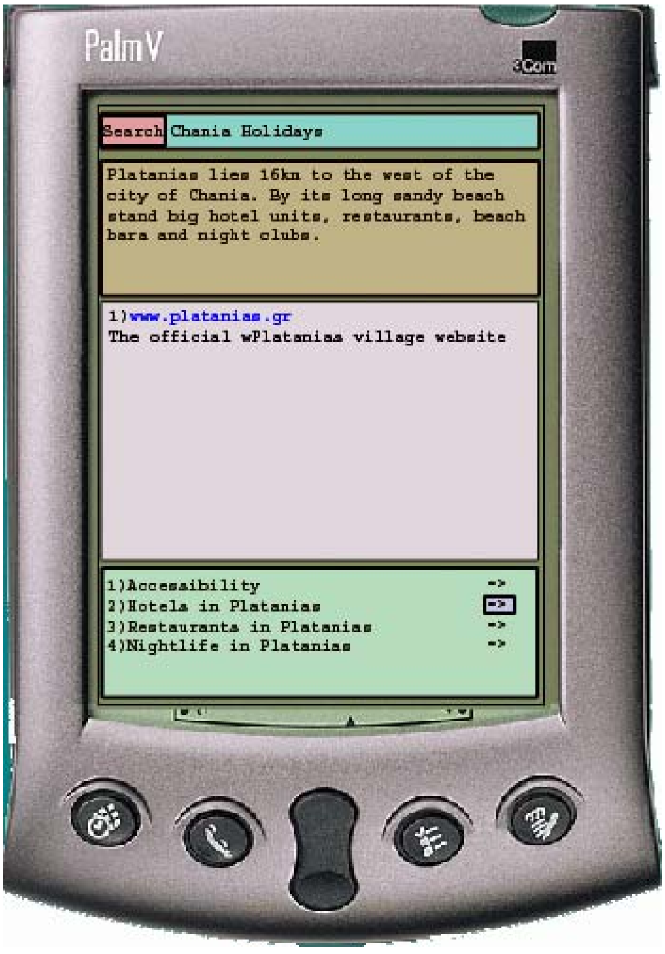


Search Chania Holidays

The Capital of the prefecture of Chania is the city of Chania. In summer, many waterfronts like Platanias, Paleochora and Sfakia attracht thousends of tourists every year.

1)www.chania.gr
The official website of the muncipality of Chania

1)City of Chania ->
2)Travel Guides of Chania ->
3)Platanias ->
4)Paleochora ->
5)Sfakia ->

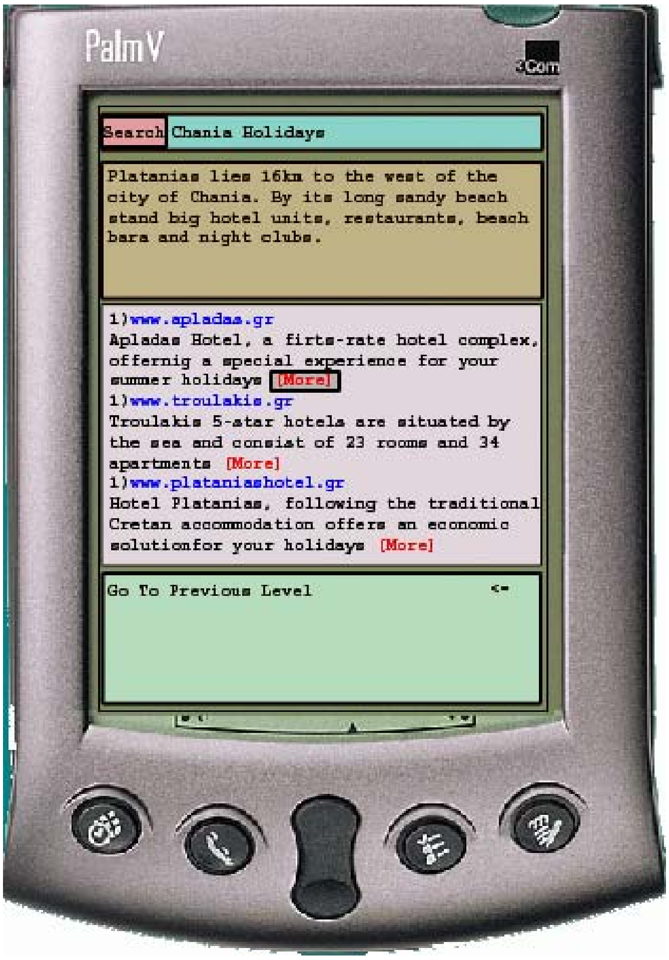


Search Chania Holidays

Platanias lies 16km to the west of the city of Chania. By its long sandy beach stand big hotel units, restaurants, beach bars and night clubs.

1)www.platanias.gr
The official wPlatanias village website

1)Accessibility ->
2)Hotels in Platanias ->
3)Restaurants in Platanias ->
4)Nightlife in Platanias ->

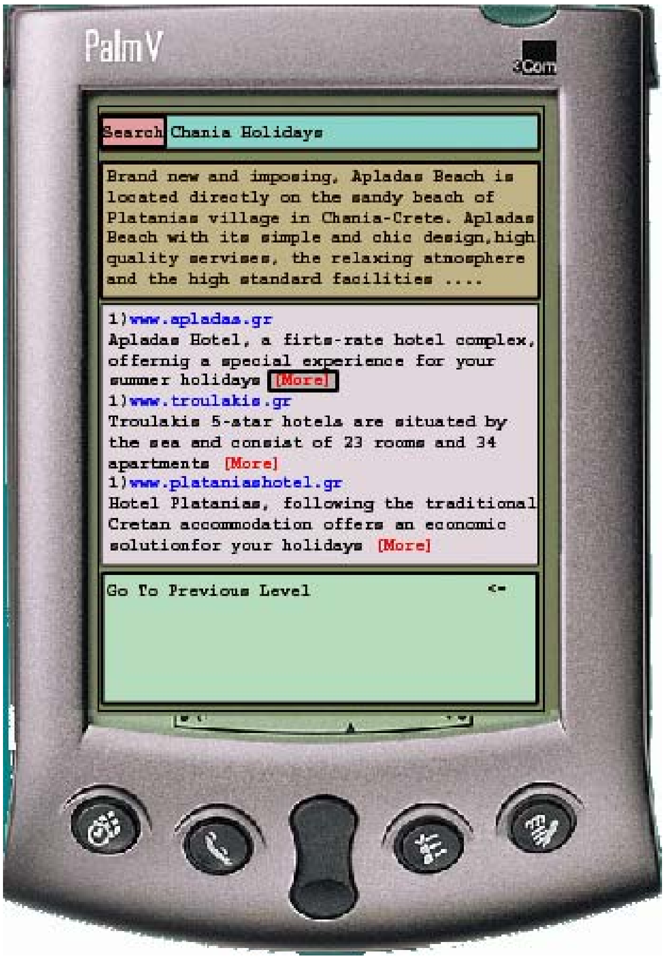


Search Chania Holidays

Platanias lies 16km to the west of the city of Chania. By its long sandy beach stand big hotel units, restaurants, beach bars and night clubs.

1)www.apladas.gr
Apladas Hotel, a firts-rate hotel complex, offernig a special experience for your summer holidays [More]
1)www.troulakis.gr
Troulakis 5-star hotels are situated by the sea and consist of 23 rooms and 34 apartments [More]
1)www.plataniashotel.gr
Hotel Platanias, following the traditional Cretan accommodation offers an economic solutionfor your holidays [More]

Go To Previous Level <-



Chania Holidays

Brand new and imposing, Apladas Beach is located directly on the sandy beach of Platanias village in Chania-Crete. Apladas Beach with its simple and chic design,high quality servises, the relaxing atmosphere and the high standard facilities ....

1)www.apladas.gr
Apladas Hotel, a firts-rate hotel complex, offernig a special experience for your summer holidays [More]
1)www.troulakis.gr
Troulakis 5-star hotels are situated by the sea and consist of 23 rooms and 34 apartments [More]
1)www.plataniashotel.gr
Hotel Platanias, following the traditional Cretan accommodation offers an economic solutionfor your holidays [More]

Go To Previous Level <=

Figure 1.6: Example Use Case 3

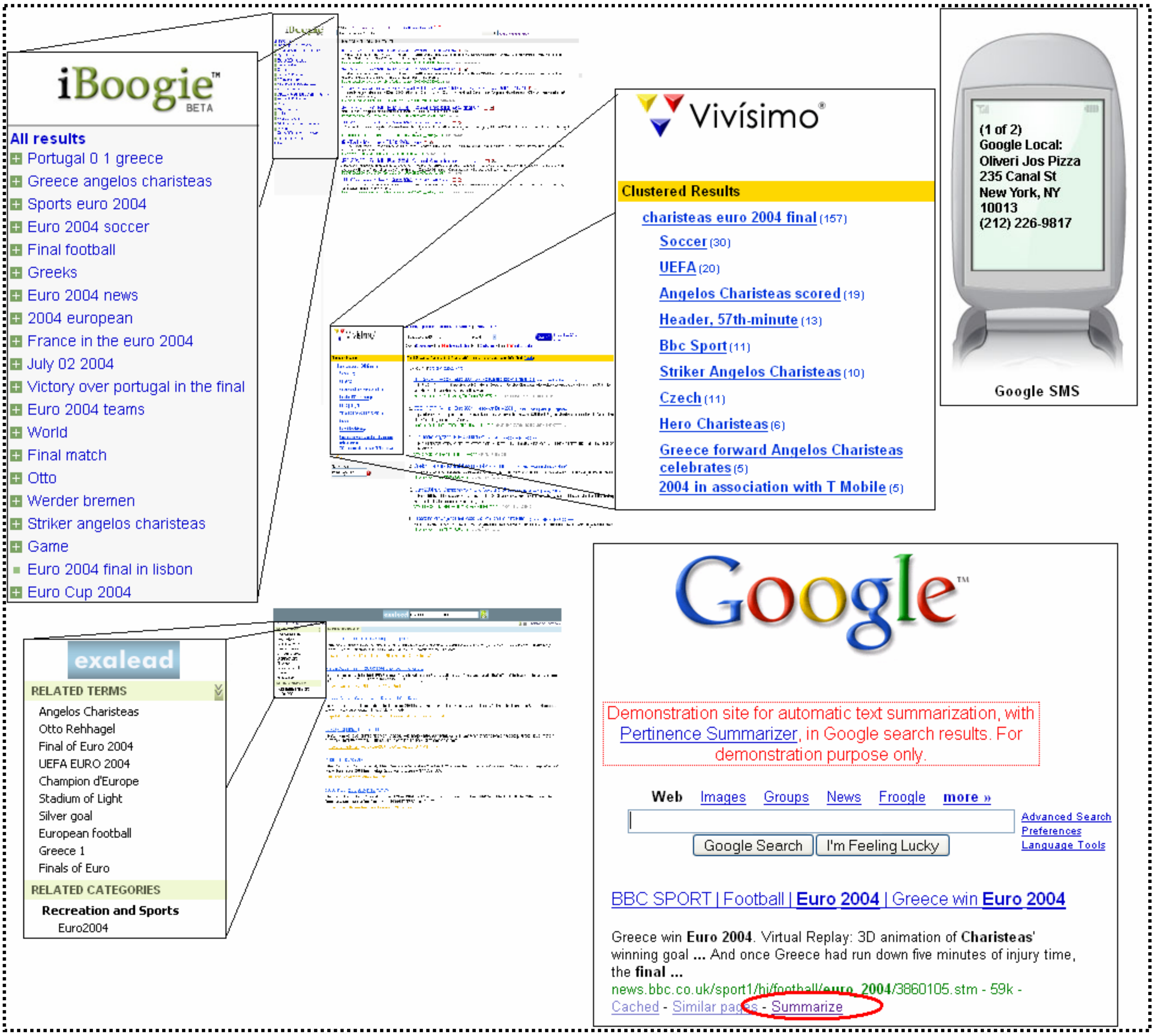
1.2 Related Work

Within the fields of interest surrounding our application, a lot of related work can be found. Regarding the multimodal part, it is a continuously developing part of both research and business. That is due to the shifting of the purchasing public into more flexible devices that usually promote their multimodal support. Narrowing it to search engines, apart from the traditional ways of I/O, there are examples of alternate I/O applications such as Google’s “Google SMS” (using text input and output – figure 1.7) or Conversay’s VoiceSurfer (for voice input/output).

Search engines and their characteristics is an even more focalized field. It’s the fact that search engines have proven to be the most important tool for browsing the web and finding specific information that makes it so “hot” a field. Focusing on to the clustering feature, it seemingly becomes a characteristic of more and more search engines. Many examples are available on the web of such search engines either using pre-defined clusters or result content created. Some of them include Vivisimo, iBoogie, Exalead and Carrot[2]. Figure 1.7 also shows the clusters created for some of the above engines, following a query (“charisteas euro 2004 final”). As for summarization, it is more or less used in every search engine at some point. However, standalone summarizers can offer this functionality over search engines as well. An example of the Pertiner’s Summarizer over Google results is also shown in figure 1.7.

1.4 Summing Up

After we’ve seen the related work on the topics of our interest, and set the base of the design for our own application, we can start implementing the back-end, namely all modules of the system that stay hidden to the user. Our implementation is a constituent of the desired outcome and the concessions we had to take due to the critical issues discussed in topic 1.2.1.



iBoogie BETA

All results
- Portugal 0 1 greece
- Greece angelos charisteas
- Sports euro 2004
- Euro 2004 soccer
- Final football
- Greeks
- Euro 2004 news
- 2004 european
- France in the euro 2004
- July 02 2004
- Victory over portugal in the final
- Euro 2004 teams
- World
- Final match
- Otto
- Werder bremen
- Striker angelos charisteas
- Game
- Euro 2004 final in lisbon
- Euro Cup 2004

Vivísimo

Clustered Results

charisteas euro 2004 final (157)
- Soccer (30)
- UEFA (20)
- Angelos Charisteas scored (19)
- Header, 57th-minute (13)
- Bbc Sport (11)
- Striker Angelos Charisteas (10)
- Czech (11)
- Hero Charisteas (6)
- Greece forward Angelos Charisteas celebrates (5)
- 2004 in association with T Mobile (5)

(1 of 2)
Google Local:
Oliveri Jos Pizza
235 Canal St
New York, NY
10013
(212) 226-9817

Google SMS

exalead

RELATED TERMS

Angelos Charisteas
Otto Rehhagel
Final of Euro 2004
UEFA EURO 2004
Champion d'Europe
Stadium of Light
Silver goal
European football
Greece 1
Finals of Euro

RELATED CATEGORIES

Recreation and Sports
Euro2004

Google

Demonstration site for automatic text summarization, with Pertinence Summarizer, in Google search results. For demonstration purpose only.

BBC SPORT | Football | Euro 2004 | Greece win Euro 2004

Greece win Euro 2004. Virtual Replay: 3D animation of Charisteas' winning goal ... And once Greece had run down five minutes of injury time, the final ...
news.bbc.co.uk/sport1/hi/football/euro_2004/3860105.stm - 59k -
Cached - Similar pages - Summarize

Figure 1.7: Related Work Examples

PART I

THE BACK-END

In this part we deal with the back-end of our application. This includes the indexing procedure, the clustering mechanism and the summarization techniques.

Chapter 2: Regarding the indexing procedure, we explain how we gathered our repository of web pages as well as the way that all necessary attributes were extracted in order to create the index (database). Finally we present a combination of HITS algorithm with the Vector Space Model, which we used in order to implement the search and ranking mechanism.

Chapter 3: The presentation of the clustering engine shows our use of a hierarchical K-Means algorithm so that we create a tree structure of all the relevant results (that a search action has returned) and their container clusters.

Chapter 4: In this chapter, we bring forward all the summarization techniques that were developed based on our needs of different types of summaries and demonstrate how they deal with our application constraints.

Chapter 2

INDEXING

2.1 Intro

The first step in order to create our application is to have the basic functionality of a search engine. To do so we can either retrieve all query-relevant documents from an existing search engine or build our own. The first case may offer more quality in the retrieved results due to its huge amount of cached websites, but it produces a bandwidth load as well as time delay to the application server. On the other hand, creating our own search engine we no longer have those problems but the quality of results is significantly lower (figure 2.1). However, after taking under consideration the following facts:

1. Bandwidth and system load is a more crucial factor.
2. The cost of implementation is low since we don’t need more than the essential functionality.
3. For creating a test application, a small webpage repository would not be that big a drawback.

we have selected to include a search engine in our system and not use an external one.

In general the architecture of a search engine is the one showed in the following picture (figure 2.2). However, we can divide the process of creating a search engine in the following 3 steps:

a) Retrieving a number of web pages and storing them locally.
b) Extract information from those web pages and create a database.
c) Implement a search mechanism in order to extract query-relevant documents from the database.

Existing Search Engine	Own Search Engine
1) Better result quality (and quantity) → Better QoS	1) Pre-processing of results available (since they are locally stored) 2) Lightweight application (we can implement only the features we need) 3) No bandwidth load (except when creating the database)
1) Bandwidth load in order to retrieve relevant web pages 2) Time delay for on-the-fly processing of retrieved results 3) System load – I/O load on the application server when many parallel searches.	1) Smaller database results in less 'good' results which decrease the quality of the service 2) Cost of implementation

Figure 2.1: Existing/Own Search Engine Pros and Cons

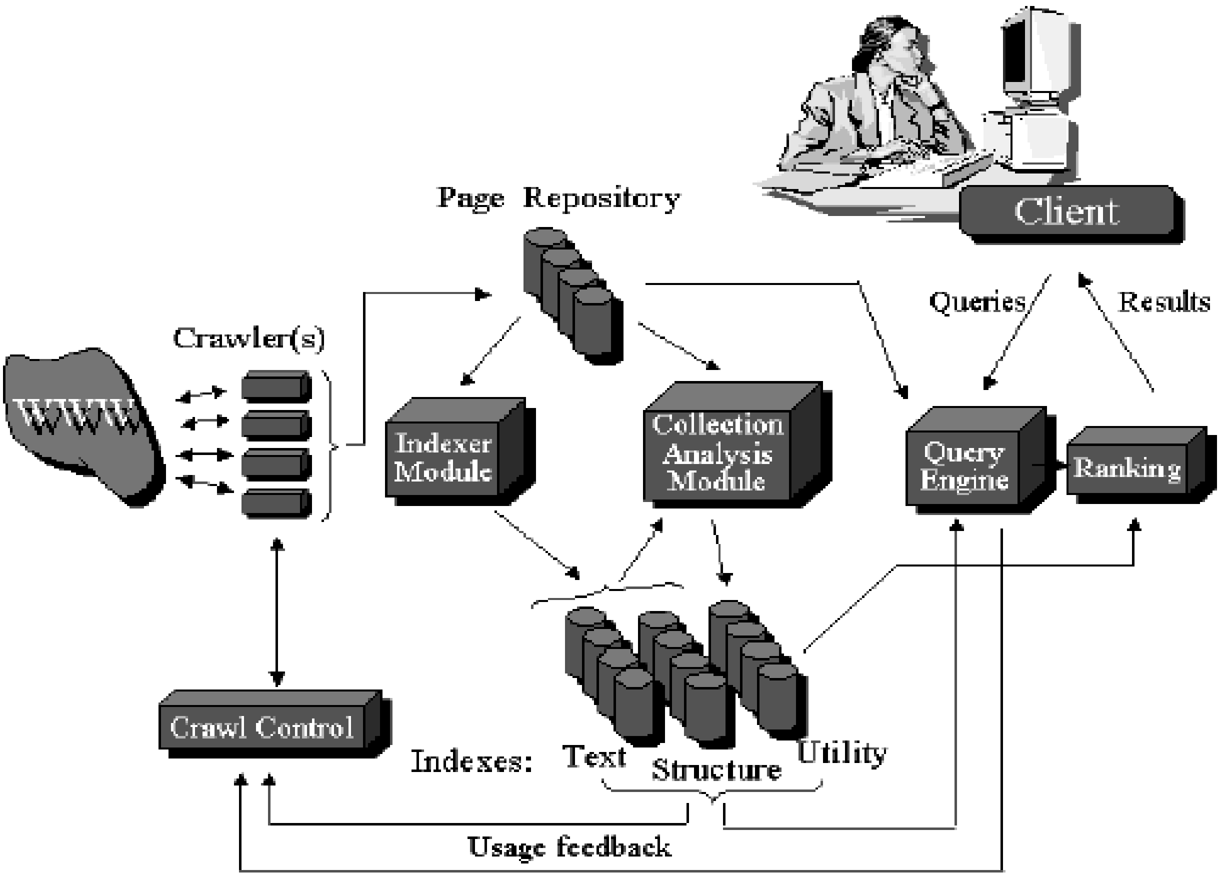


Figure 2.2: Architecture of a Search Engine.

2.2 Creating the repository

Before starting crawling websites to create our repository we must define what kind of web pages we want. Since the amount of web pages can't be really big we have to choose characteristic websites for the whole internet. Therefore we have selected the following two sites: "*www.cam.ac.uk*" and "*www.thefa.com*".

The first one is the University of Cambridge web site. Including web pages of numerous different departments, plus user home pages, this website can be regarded as a miniature of the whole web with both well structured and random content. Moreover its content is more or less distributed in relatively few topics (having to do with academic issues), which means one can find similarities and so clustering has a meaning.

The second website is the official English Football Association website. Contrary to the Cambridge website, this has a specific news site structure, with its pages being in fact articles. This makes evaluating our summarization techniques more easy and accurate.

As for the number of web pages, we have stored about 10000 from the first website and another 10000 from the second. This makes a total of 20000 web pages, which can be considered enough for our demo application.

Having selected which websites to crawl, we used Larbin [23], an open source crawler, in order to get them. A crawler is a small program that retrieves pages from the web following the links between them. Its operation can be shortly described like this:

1. The crawler starts with an initial set of URLs which it places in a queue
2. Prioritizes queue items according to the crawler control module.
3. Gets the next queue URL, and downloads it into the page repository.
4. Puts all new URLs, the just downloaded page pointed to, in the queue.
5. Continues this procedure until it reaches its stop condition.

In particular we configured Larbin to download only web pages (*.html, *.htm files) just from the sites mentioned above and not to follow external links. We also adjusted Larbin so that it would descend up to 10 levels down each site.

After our crawler has selected the desired amount of web pages, those pages were stored in an internet-address-like directory structure and with no further processing we were able to extract information from them and create our index.

2.3 Creating the index

The need to create a database, for the amount of the web pages we have in our repository, lies to the fact that a certain pre-processing is necessary in order to extract and rank query relevant results. Moreover it offers a quicker response and significantly less I/O for each query.

Both the algorithms and the source code we used for indexing, searching and ranking are based on the academic work of Epimenides Voutsakis [5]. Our contribution was to make all the necessary adjustments in so that it would be fit for our application. The source code is written in Perl programming language and a BerkleyDB database is used as our index.

The search-ranking algorithm (which we explain in the following topic) is a combination of classic information retrieval methods and link analysis methods. Therefore it is required that we extract both link related information and term counts. We use the word 'term' instead of 'word' to emphasize the use of a stemmer so that we keep stems, not words. Moreover a list of stop-words (commonly occurred words, unlikely to give useful information) is excluded from our index in means of optimization. In particular, the attributes that we store in our index are the following:

- Filename (internet address), File ID, Title
- Forward links, Backward links (1)
- An inverted file for every term (2)
- A normalized term frequency for each web page (3)

Finally an inverse document frequency (4) index is created after all the other indexes have been created. Regarding (1), (2), (3), (4) and how they are used, a detailed analysis follows in the next topic.

We must note that this whole procedure needs to be done only once at the beginning since whenever a new set of pages is inserted in the repository; an ‘add’ function can index just the newly added pages. The only exception is the creation of the inverted file which must be regenerated.

2.4 Implementing Search Functionality

As we have mentioned before, we only need our search engine to have basic functionality. In other words, all we want it to do is to return ranked relevant results following a query. Therefore, there should be a searching function that finds relevant results and a ranking function that sorts them by terms of relevancy.

For the searching function all that is used is the inverted file index. The inverted file is responsible for keeping track of all the web pages a term appears in. The algorithm we introduce to do this job is actually very simple. Specifically, after defining that a relevant result is a web page that includes all terms of the query, we just have to find which web pages satisfy that rule.

The algorithm below shows the steps of the searching procedure:

1. Break query in its terms.
2. For every term search the inverted file to find all the web pages it is in.
3. Export a web page as an accepted result if it includes all terms of the query.

After we gather all these results we have to sort them using the ranking algorithm. The ranking algorithm combines Vector Space Model features as well as the HITS ranking method. Before we analyze the steps of our algorithm, a short presentation of their theoretical background is presented.

2.4.1 Vector Space Model

The Vector Space Model (introduced by Salton and associates [4]) is a way of representing documents through the words that they contain. It is considered a classic information retrieval (IR) method and it is used mostly for ranking (or measure similarity between) offline documents. By the word "classic" we mean IR that is based solely on the words in the documents, without taking in mind the information such as the one that links between web pages provide.

The Vector Space Model's main characteristics are enlisted bellow:

- The VMS considers a high-dimensional vector space with one dimension per term.
- Each document or query is represented as a term vector in this vector space.
- Entries of terms occurring in the document are positive, else they are zero.
- The similarity between a document and a query (or another document) is usually computed by the dot-product (cosine measure) of their term vectors.
- The ranking of documents is based on the similarity score of each document for the given query.

However, some features have been gradually added in order to deal with the following matters:

a) Not all words are equally useful. A word is more likely to be highly relevant in a document A if i) it is frequent in document A and ii) infrequent in other documents.

Instead of using the exact number of times a term appears in a document, this value is normalized by dividing it by the maximum frequency of any term in the document. This attribute is called “normalized term frequency” and defined by the next assignment:

$$tf = \frac{\text{frequency of a term in a documet}}{\text{maximum frequency of any term in the document}}$$

In fact tf could be considered as a measure of how well a term describes a document. By using tf, it also stops the event of larger documents scoring higher.

b) Generally rare words are usually more important than frequent words.

In order to make rare terms more important than the common ones, the use of Inverse Document Frequency (idf) of a term is proposed. The idf of a term **i,** is given by the following rule:

$$idf_i = \log \frac{N}{n_i},$$

where **N** is the number of documents and **n$_i$** is the number that contain term **i**.

The tf-idf weighting scheme arises by a further multiplication of the values of tf and idf to create the new ranking score.

To sum up, the Vector Space Model processes term frequencies in documents in order to calculate similarity and to rank relevant documents, producing better results for small and coherent collections like digital libraries, journal articles, newspapers. However, the absence of handling the information derived from web related attributes such as links between pages makes it less useful for internet content. Thus, a link analysis ranking method such as HITS can be complemental to it, in search for better ranking techniques.

2.4.2 The HITS Algorithm

Internet is a massive, distributed, incoherent, continuously updating information collection; therefore classic information retrieval methods are insufficient. The solution rises from exploiting linkage among web pages, and this lead to the development of algorithms like HITS and PageRank [3]. The main idea behind them is that a link from page A to page B can be considered a recommendation of page B by the author of page A

HITS (Hyperlink-Induced Topic Search) [7] is a query dependent ranking technique. For every webpage, two scores are calculated: an *authority* score and a *hub* score. The authority score tells us how likely it is that the webpage is relevant to the query. The hub score informs us if the webpage points (has links) to many authorities. This separates web pages into authorities and hubs but not distinctly since hubs may be authorities too. In practice, the ranking algorithm invokes the following two conclusions:

a) Good authorities are pointed to by many hubs.

b) Good hubs point to many authorities.

In the following picture (figure 2.3) we demonstrate an example of how result web pages are linked together. As we can see, web pages C and F can be considered "good" hubs as they are pointing to many of the results. Moreover, pages A, H and C are rather good authorities being pointed by many hubs. The HITS algorithm shall produce a higher score for those web pages.

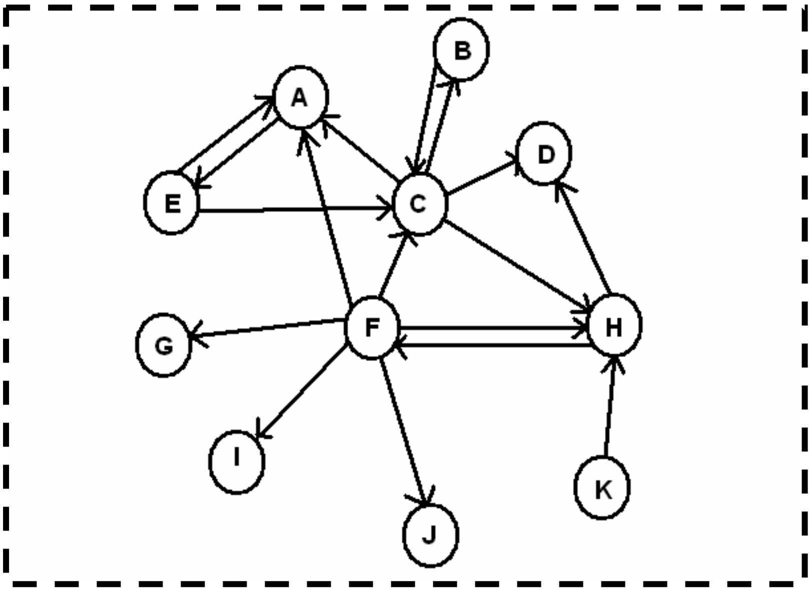


Figure 2.3: Example of linkage between results.

In order to calculate HITS scores of the results, firstly the focused subgraph **S** of the query is generated. The focused subgraph is the expanded set of results and their forward and backward links. This process is done in the steps below:

1. A set **R** of relevant to the query results is created.
2. **S** is currently the set **R** (alternatively only **t** pages from R, where **t** is a designer defined parameter)
3. For every page **p** ∈ **R**:
    a. Include all pages that **p** points to (forward links) in **S.**

   b. Include, up to **d**, pages that point to **p** (backward links) in **S** (where **d** is a designer defined parameter)

4. The graph induced by **S** is now the focused subgraph of the query.

After the focused subgraph is created, the Link Analysis phase of HITS takes place. Every page **p** in **S** is associated two scores. A hub score **h(p)** and an authority score **α(p)** which are initially set to the value of 1. Defining that **p→q** means that “page **p** has a hyperlink to page **q**”, the authority and hub scores are calculated by the following rules:

$$\alpha(p) = \sum_{q \to p} h(q)$$

$$h(p) = \sum_{p \to q} a(q)$$

This operation is repeated until it converges to a stable set of scores. Moreover, after each iteration, the scores are normalized. Figure 2.4 below shows the steps of the algorithm.

```
1) Initialize α(p), h(p) to 1 for every page p.
2) For every page p, repeat until convergence:
      a. Set $\alpha(p) = \sum_{q \to p} h(q)$
      b. Set $h(p) = \sum_{p \to q} a(q)$
      c. Normalize α(p) and h(p) values.
3) End
```

Figure 2.4: HITS algorithm link analysis steps

2.4.3 Our Approach

The ranking function we use takes advantage of both classic IR and link analysis features. A variation of the Vector Space model is used as the classic IR method. This offers us additional weight in the score produced by the HITS algorithm we used as our link analysis method.

The reasons we chose HITS as our link analysis methods have to do with the nature of our application and are presented below:

- For broad topics, HITS produces stable, robust communities despite starting from a very small sample of relevant pages in the initial root set. This helps our soon to create clusters to be distinct from each other.
- On narrowly focused topics, HITS often returns good resources for a more general topic. This ‘topic generalization’ allows the automatic characterization of certain topics; witch is also helpful both to the clustering engine and the end user receiving the results.
- It is hard to mislead by spamming.

Going on to explain our version of the ranking algorithm we must firstly remind that the searching algorithm has already given us a set of results (those who contain all terms from the query). This set forms the initial set **R** used by HITS in order to create the focused subgraph. Then all forward and backward links are added to create the focused subgraph.

The link analysis phase is the exact one of HITS. We have selected a total of 20 iterations; at witch convergence of the authority and hub scores of each page has been achieved. The normalization function is the typical one, so what we do is apply the following rules after each iteration:

$$\sum_p (a(p))^2 = 1 \quad \text{and} \quad \sum_p (h(p))^2 = 1$$

The authority score of each page is one of the factors in the product that produces the final score of each page. The other factors emerge from the Vector Space Model. In particular, they are the tf and idf values of the terms in the query. Instead of the original tf and idf rules, we declare those values in a slightly different way. The term frequency value of a term **i** in a document is given by the rule:

$$tf_i = \frac{\log\left(n_i + 1\right)}{\log\left(n\right)}$$

where **tf$_i$** is the term frequency, **n$_i$** is the number of times that term **i** appears in the document and **n** is the length of the document in terms.

The idf value **idf$_i$** of a term **i** is given by the rule:

$$idf_i = \log\left(\frac{N}{N_i}\right)$$

where **N** is the total number of pages and **N$_i$** the number of pages that term **i** appears in.

The way they contribute to the final score goes like this:

a) Firstly, the query Q is broken into its terms

b) For every term t a product $tf_t * idf_t$ is calculated

c) All these products are summed together and this makes the other factor of our final score.

Summing up, the final rule that creates the score for each result is the following one:

$$S(p) = a(p) * \sum_{t \in Q} tf_t * idf_t \Rightarrow$$

$$\Rightarrow S(p) = a(p) * \sum_{t \in Q} \frac{\log(n_i + 1) * \log\left(\frac{N}{N_i}\right)}{\log(n)}$$

After every result webpage is assigned its score, the list is sorted in decreasing order. Along with the score for each page, we extract its title (why we do that at this point will be explained in the next chapter). Since there is no need for further functionality in our search engine, we are ready to proceed with creating the cluster hierarchy of the results. Figure 2.5 shows an example list of web pages returned by our search engine for the query “charisteas euro 2004 final”.

```
1.36631768703335 http://www.thefa.com/Euro2004/NewsAndFeatures/Postings/2004/06/MinbyMin_GreeceCzech.htm
        TheFA.com - Dellas heads Greece through
1.26777082661733 http://www.thefa.com/Euro2004/Teams/Postings/2004/04/TeamProfile_Greece.htm      TheFA.com - Greece
1.14618186557254 http://www.thefa.com/Euro2004/NewsAndFeatures/Postings/2004/06/MatchReport_GreeceCzech.htm
        TheFA.com - Greek tragedy for Czechs
0.984129452227208         http://www.thefa.com/Euro2004/NewsAndFeatures/Postings/2004/07/MR_PortugalGreeceFinal.htm
        TheFA.com - Greece kings of Europe
0.952574253357013         http://www.thefa.com/Euro2004/NewsAndFeatures/Postings/2004/07/Reaction_SteliosDelight.htm
        TheFA.com - Stelios&apos; Euro delight
0.86085287919561 http://www.thefa.com/Euro2004/NewsAndFeatures/Postings/2004/07/MinByMin_PortGreeceFinal.htm
        TheFA.com - Angelos the Greek god
0.766396711595953         http://www.thefa.com/Euro2004/NewsAndFeatures/Postings/2004/07/Euro2004_AllStarSquad.htm
        TheFA.com - Four All-Star Lions
0.566970605296713         http://www.thefa.com/Euro2004/NewsAndFeatures/Postings/2004/07/Final_preview.htm TheFA.com       -
Moment of truth
0.556232072426952         http://www.thefa.com/Euro2004/NewsAndFeatures/Postings/2004/07/ReactionGreeceCamaraderie.htm
        TheFA.com - Greece upset the odds
```

Figure 2.5: Example Search Results

Chapter 3

CLUSTERING

3.1 Intro

Clustering is the process of grouping similar documents together to expedite information retrieval. It can be categorized as hierarchical and non-hierarchical clustering. Hierarchical clustering distributes documents iteratively into categories and subcategories, while non-hierarchical clustering distributes them in just one level of categories. It is crucial to highlight the fact that, in comparison to document classification, these categories don't preexist, but are created by the context of the documents to cluster.

The ability to identify which documents belong to what group derives from calculating the similarity between documents. Many algorithms apply to this, usually by identifying relevant attributes of documents, determining appropriate weights for those attributes and calculating distances. Applying clustering to the web search context actually means organizing web pages into groups, so that different groups correspond to different user needs. In contrast to document clustering, web page clustering has to deal with issues like the vast amount of web pages and their constantly changing heterogeneous structure. Some of the most important clustering algorithms are: K-Means, Agglomerative Hierarchical Clustering (AHC), Suffix Tree Clustering and Self organizing Maps. The algorithm we use is a hierarchical variation of the K-Means Clustering algorithm.

The presence of clustering in the application we design is of great importance. In general, clustering web pages is a good way of making relationships between them obvious, which aids the user-application interaction. In our case, it helps us create a structure of the result pages that will 1) give the user an organized self explanatory browsing framework and 2) partition the amount of information transmitted to the user and therefore reduce bandwidth needs.

Further in this chapter, follows a deeper analysis of the clustering engine we created and the way we built our structure of web pages.

3.2 The Clustering Engine

In a few words, what a clustering engine should do in our application is, given a set of result web pages, to create a predefined number of clusters, distribute the pages into them and then distribute clusters into super-clusters to create a tree structure.

In order to do so, we used *spkmeans* [8], a clustering and dimension reduction toolkit. In particular we used *h-spkmeans* which is a hierarchical version of *spkmeans*. *Spkmeans* is written in C++ and as foretold it uses the Spherical K-Means clustering algorithm, enhanced by a technique for significantly reducing the amount of similarity computations required. This tool works, like this: it gets a word-document matrix in CCS format as input, and outputs the clusters of web pages. In order to use it, the following steps took place:

1) Firstly we built the executable programs from the corresponding source code provided.

2) We created the word-document matrix in CCS format.

3) We used the CCS matrix as input to the executable and got a file showing which page or cluster belongs to which cluster.

A closer examination of the above steps follows:

Step 1: The source code of *h-spkmeans* was available on the designer's website. Before building the executables, certain alterations were made to the source files. In particular, we changed the way the output is displayed into something easier to handle later when we create the structure. Then we compiled the source under the g++ compiler and got the executable program.

Step 2: Before we explain how we created the word-document matrix, we should indicate that it actually is a numerical representation of the web pages (obviously of the words they include). In order to build it we used a tool called *mctester*, also provided at the *spkmeans* webpage, written in C++ and compiled under g++. This tool takes a set of documents (web pages in our case) and creates their word-document matrix in CCS format. We used its default parameters, and a typical command would be:

```
./mctester -t tfn ./docdir ./outputdir
```

where the parameter "-t tfn" declares that the scaling of values is that of the inverse global term frequency with certain normalization. Path "docdir" is the directory of the result web pages, while "outputdir" is where the output files, defining the word-document matrix, will be extracted.

Step 3: Since we have both the executable and the word-document matrix, all we have to do before running the appropriate command is to specify all other necessary parameters. Those would be the branching factor and the depth of the desired hierarchical structure. The branching factor will define how many children cluster each parent cluster is allows to have, whereas the depth parameter will determine at what level of clusters the results should be assigned. According to our primal design for the application we set the rule that a maximum of 5 sub-clusters in each cluster should be available. Moreover, the user shouldn't have to descend more than three levels. Therefore, we set the branching factor to be 5 and the level parameter to be 3 (figure 3.1). This results to the following clustering command:

```
./spkmeans -l 3 -c 5 -O ./outputdir/ ./mc_outputdir
```

where the '-l' parameter is the level, '–c' the branching factor, -O the outputdir and the final parameter is the matrix path. All other parameters available by the program are set to their default values.

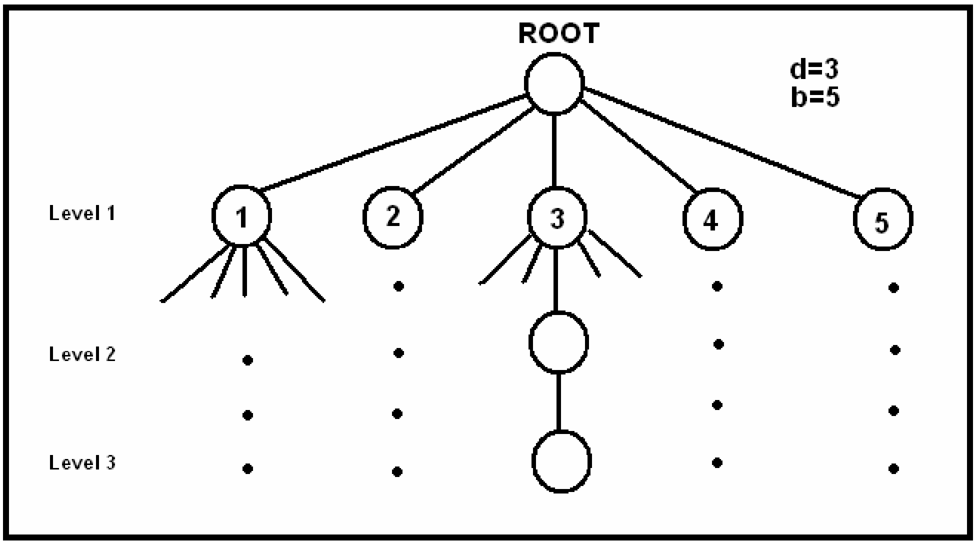


Figure 3.1: Tree Structure of Clusters

Figure 3.2 shows an example output of the *spkmeans* program, while figure 3.3 is an example of the output files we later use in order to create the tree structure.

```
Reading the _dim file...
Reading the _col file...
Reading the _row file...
Reading the _nz file...
Reading file time: 0.01 seconds.
Now clustering...
3-level hierarchical clustering

Level 0 ...
Original Size: 20

Level 1 ...
h_ClusterSize[0]=2
       .
       .
h_ClusterSize[4]=3

Branch 0 ...
       .
       .
Branch 4 ...

Level 2 ...
h_ClusterSize[0]=1
       .
       .
h_ClusterSize[24]=0

Branch 0 ...
       .
       .
Branch 23 ...

Level 3 ...
h_ClusterSize[0]=0
       .
       .
h_ClusterSize[124]=0
Hierarchical spherical k means algorithm
Final number of clusters: 125
Number of documents: 20
Number of words: 3166
epsilon: 0.001
initialization method: random perturbation
perturbation magnitude: 0.1
encoding scheme: normalized term frequency inverse document
frequency
objective function: nonweighted
computation time:
CPU Usage: user = 0 seconds 56991 ms, system = 0 seconds 3000ms
Time per iteration = 0.0284955
ELAPSED Time: 0 seconds 76264 ms
Output matrix file is:./mc_outputdir/documents
Memory consumed :250593
```

Figure 3.2: Sample Spkmeans Program Output

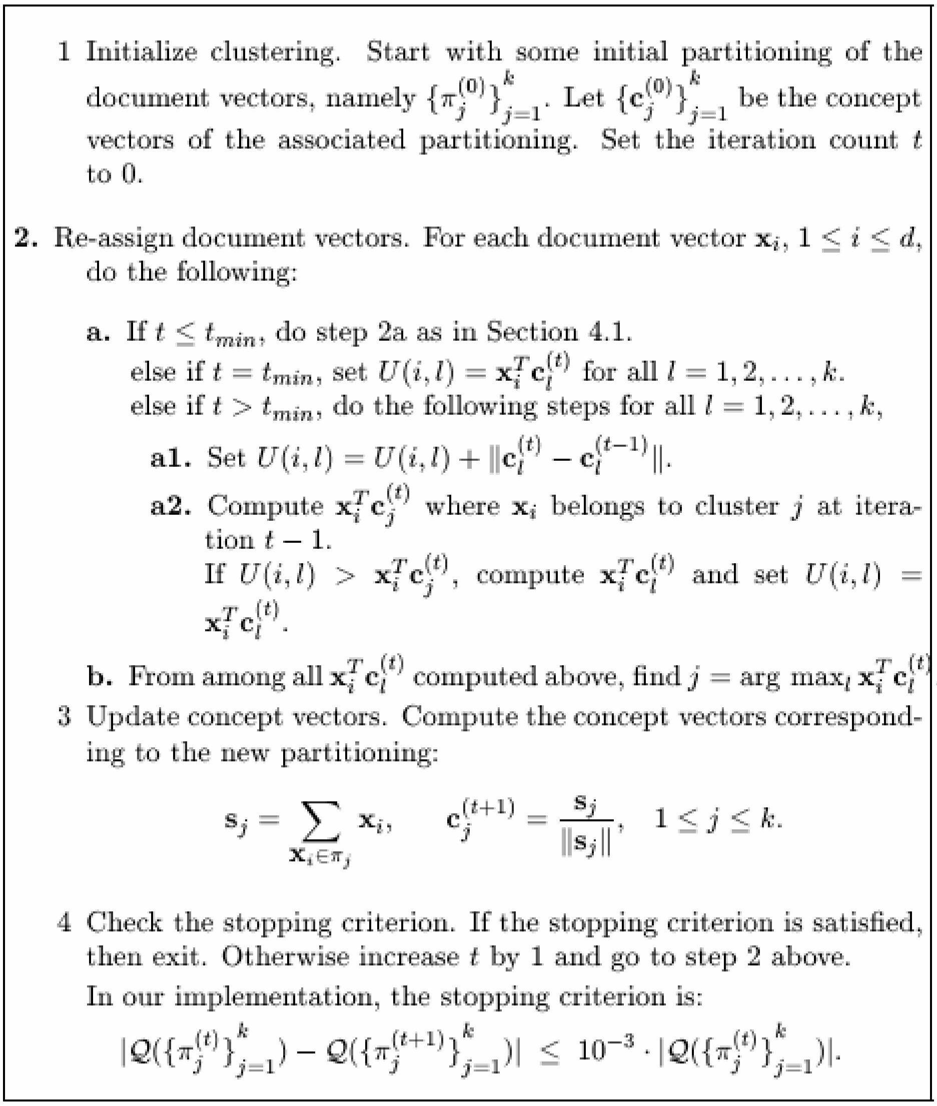
```
0 0 1 2 3 4

0 0 1 2 3 4
1 5 6 7 8 9
2 10 11 12 13 14
3 15 16 17 18 19
4 20 21 22 23 24

0 0 1 2 3 4
1 5 6 7 8 9
2
3
4
5 25 26 27 28 29
6 30 31 32 33 34
7 35 36 37 38 39
8
9 45 46 47 48 49
10 50 51 52 53 54
11
12
13
14 70 71 72 73 74
15
16 80 81 82 83 84
17 85 86 87 88 89
18
19 95 96 97 98 99
20 100 101 102 103 104
21 105 106 107 108 109
22
23 115 116 117 118 119
24
```

Figure 3.3: Sample Spkmeans Program Output File

Previous to proceeding with creating a structure of the clustered results, we adduce a short theoretical analysis regarding the clustering algorithm used by *spkmeans*.

3.2.1 Hierarchical Spherical K-Means Algorithm

Spherical K-Means [9] is an algorithm that exploits the sparsity of the data while giving meaningful results at the same time. The documents are represented as vectors in the vector space. Then, clustering is the partitioning of the document collection into the disjoined subsets $\pi_{1,}$ π_2... π_k. The decision of which cluster a document belongs to or how “good” or “coherent” a cluster is, derives from cosine similarity measures and the creation of concept vectors for a cluster (its normalized centroid). Figure 3.4 shows the steps of the Spherical K-Means algorithm:



```
1 Initialize clustering. Start with some initial partitioning of the
  document vectors, namely $\{\pi_j^{(0)}\}_{j=1}^{k}$. Let $\{\mathbf{c}_j^{(0)}\}_{j=1}^{k}$ be the concept
  vectors of the associated partitioning. Set the iteration count $t$
  to 0.

2. Re-assign document vectors. For each document vector $\mathbf{x}_i$, $1 \le i \le d$,
   do the following:

   a. If $t \le t_{min}$, do step 2a as in Section 4.1.
      else if $t = t_{min}$, set $U(i,l) = \mathbf{x}_i^T \mathbf{c}_l^{(t)}$ for all $l = 1, 2, \ldots, k$.
      else if $t > t_{min}$, do the following steps for all $l = 1, 2, \ldots, k$,
         a1. Set $U(i,l) = U(i,l) + \|\mathbf{c}_l^{(t)} - \mathbf{c}_l^{(t-1)}\|$.
         a2. Compute $\mathbf{x}_i^T \mathbf{c}_j^{(t)}$ where $\mathbf{x}_i$ belongs to cluster $j$ at itera-
             tion $t-1$.
             If $U(i,l) > \mathbf{x}_i^T \mathbf{c}_j^{(t)}$, compute $\mathbf{x}_i^T \mathbf{c}_l^{(t)}$ and set $U(i,l) =$
             $\mathbf{x}_i^T \mathbf{c}_l^{(t)}$.

   b. From among all $\mathbf{x}_i^T \mathbf{c}_l^{(t)}$ computed above, find $j = \arg\max_l \mathbf{x}_i^T \mathbf{c}_l^{(t)}$.
3 Update concept vectors. Compute the concept vectors correspond-
  ing to the new partitioning:

      $\mathbf{s}_j = \sum_{\mathbf{x}_i \in \pi_j} \mathbf{x}_i, \quad \mathbf{c}_j^{(t+1)} = \frac{\mathbf{s}_j}{\|\mathbf{s}_j\|}, \quad 1 \le j \le k.$

4 Check the stopping criterion. If the stopping criterion is satisfied,
  then exit. Otherwise increase $t$ by 1 and go to step 2 above.
  In our implementation, the stopping criterion is:
  $|\mathcal{Q}(\{\pi_j^{(t)}\}_{j=1}^{k}) - \mathcal{Q}(\{\pi_j^{(t+1)}\}_{j=1}^{k})| \le 10^{-3} \cdot |\mathcal{Q}(\{\pi_j^{(t)}\}_{j=1}^{k})|.$
```

Figure 3.4: Spherical K-Means Algorithm

Important Features

- A careful selection of initial partitions is important

- The total time complexity for τ iterations is $O(nz*k*\tau)$, where nz is the number of non-zero entries in the sparse matrix and k the number of clusters.
- The similarity estimation variation used by *spkmeans* (step 2 of the algorithm) reduces the computational bottleneck of the dot product computations between all document vectors and concept vectors.
- The hierarchical tag points to the fact that this procedure is done not only for documents, but for clusters as well.

Since Spherical K-Means algorithm was just used and not implemented or changed in our work, we reckon that a deeper analysis wouldn't be necessary at this point. Anyone, interested in a more in depth presentation should follow the given references.

3.3 Building the Structures

Now that we have the clusters and the web page they contain, it is necessary to create a structure containing this information so that it can be accessible from our front-end. We have decided that an easy way to do so is to store it in an xml file. Besides that, this xml file could contain webpage related information such as its score and title (both of which are available from the search engine) and that would be of help as well.

In fact, the xml file will be a depth first traversal of the tree structure of clusters. Thus, it is vital that we transform the webpage-cluster correspondence, which the clustering engine has extracted, into an actual tree structure. The source code of the program we developed for doing so is written in C++ programming language. In this topic we exhibit the implementation and analyze its features.

Firstly, we must indicate that in order to create the tree structure we use lists. The main classes we implement are those of a Document and a Cluster. Moreover DocumentList and ClusterList classes are defined as the lists for the previous classes.

Class Document contains information about each webpage. In particular, it contains its name, its score and the cluster it belongs to. It also contains a potential pointer to another Document, which is needed when it is inserted in a DocumentList. Available methods provide nothing more than the basic functionality of extracting the Document's attribute values.

Class Cluster contains information such as its name, its score, the level it belongs to, the number of results it contains and again a pointer to another Cluster in order to act as a ClusterList node. Again the basic methods are implemented, to retrieve its attributes' values.

Class DocumentList is the implementation of a list of Documents. It contains the number of elements in the list plus two pointers at the first and last element of its method use. The methods it has are those of insertion and removal. Insertion is always done by putting the new element at the end of the list. The deletion depends on a Document's name, so a 'seek' method is also added in order to find that Document. Finally, a method that keeps just the five highest scoring Documents is implemented. This method is necessary when a DocumentList has more than five elements, so in order to present them later to the user we have to reduce its number; that is done by removing the "worse" Documents.

Our final class, ClusterList is similar to the previous class. It has as well a number of elements attribute and pointers to the first and last Cluster element. Apart from the basic methods, only an insert method is needed and this is done again by adding the new element at the end.

After we explained the implemented classes, we can now show how by using them the structure is created. Firstly a Document instance is created for every web page in the search engine output. All those Documents are inserted in a DocumentList, which now contains all results. Then, a Cluster instance is created for every cluster the clustering engine output contains. Every level of our tree structure is created as a ClusterList of those Clusters that have the corresponding level attribute value. This final structure is shown in figure 3.5 below:

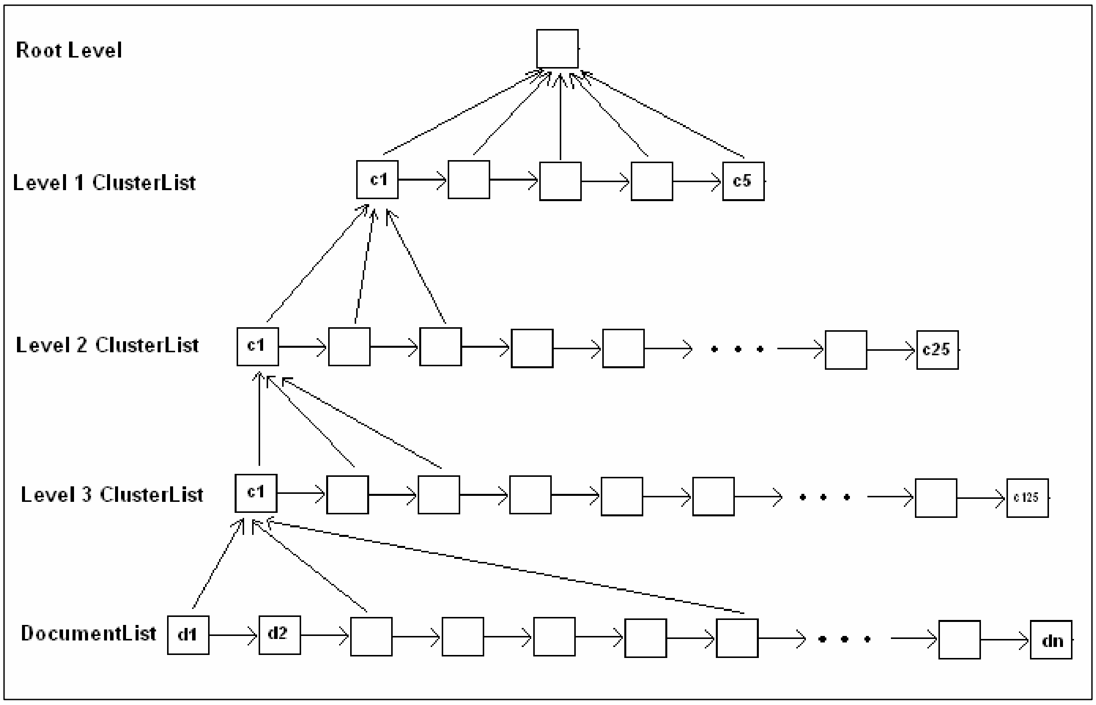


Figure 3.5: Tree Structure using Lists

The creation of the whole structure was implemented by applying the necessary functions on the classes we have defined. After that, the only thing we have to do in order to create the desired xml file, is print this structure in a specific format. However, during the printing procedure, certain actions must take place, and therefore some additional functions are as well implemented. The most important of those actions involve the calculation of a cluster's score, and the selection of the best results for a level 3 Cluster.

In the first case, what we did in order to get a score for a cluster was to sum up the scores of all web pages (hierarchically) belonging to the specific cluster and then divide that sum with the total number of pages in that cluster:

$$Score_C = \frac{\sum_{wp \in C} Score_{wp}}{\sum_{wp \in C} wp},$$

where C the cluster, $Score_C$ its score, wp a webpage and $Score_{wp}$ its score

As for selecting the best results, we first create a DocumentList of all Documents belonging to that cluster and then we take advantage of the method that keeps the five highest scoring Documents, which is available for DocumentList elements (figure 3.6).

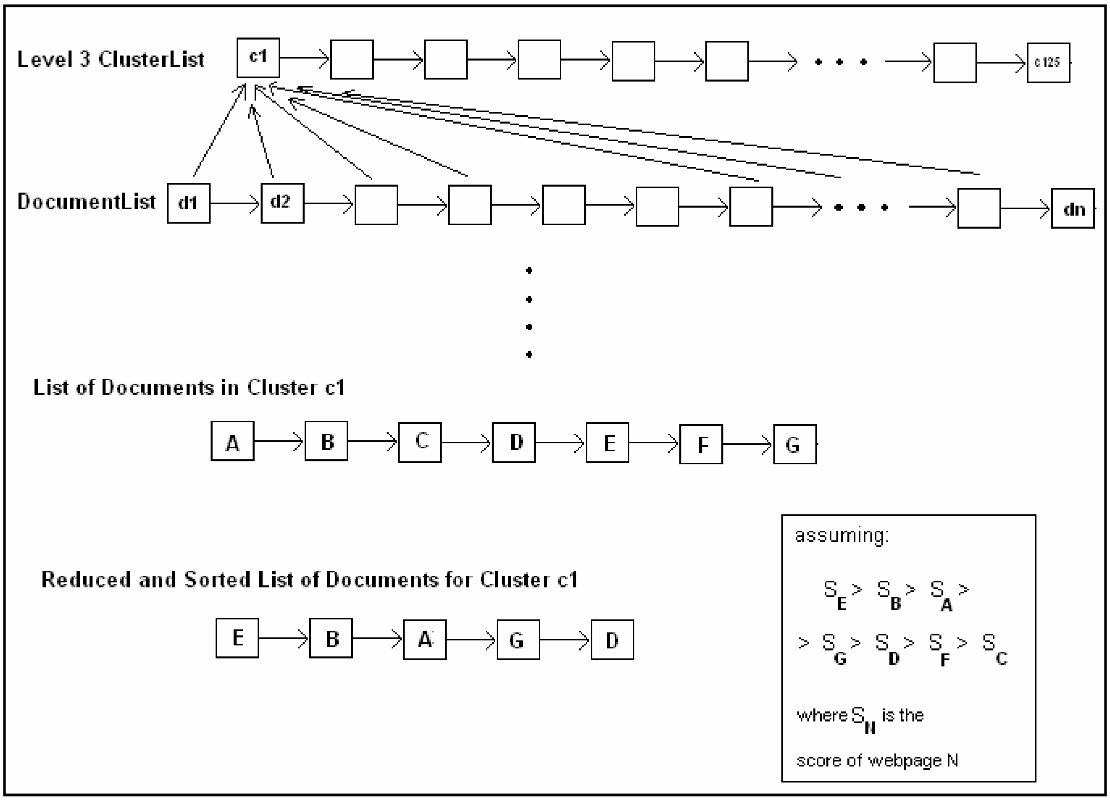

Level 3 ClusterList
c1
c125
DocumentList
d1
d2
dn
List of Documents in Cluster c1
A
B
C
D
E
F
G
Reduced and Sorted List of Documents for Cluster c1
E
B
A
G
D
assuming:
$S_E > S_B > S_A >$
$> S_G > S_D > S_F > S_C$
where S_N is the score of webpage N

Figure 3.6: DocumentList for a Result Cluster

Regarding the printing to an xml file procedure, we just dump all the above information in the format shown in figure 3.7 below. To do so we iteratively traverse the tree starting from the root Cluster and descending using a depth first display function.

```
<?xml version="1.0"?>

<Root name="root">
 <Level1 name="1Level_Cluster0">
  <Results> 111 < /Results>
  <ClusterScore> 2.542988 < /ClusterScore>
  <Level2 name="2Level_Cluster0">
   <Results> 11 < /Results>
   <ClusterScore> 1.242591 < /ClusterScore>
   <Level3 name="3Level_Cluster0">
    <Results> 3 < /Results>
    <ClusterScore> 0.395571 < /ClusterScore>
    <Result name="A result">
     <Title>The Result's Title< /Title>
     <Score> 0.611093 < /Score>
    < /Result>
    .
    .
    .
   < /Level3>
   .
   .
   .
  < /Level2>
  .
  .
  .
  <Description> To be added by the summarization module < /Description>
 < /Level1>
 .
 .
 .
< /Root>
```

Figure 3.7: Sample XML File Format

Summing up, we now have an almost final version of the xml file we use in our front-end. Having information about the structure of clusters, the scores and titles of web pages and the scores of clusters, the only thing missing is a really short description of the first level clusters; this description is added by the summarization module, that we discuss in the next chapter.

Chapter 4

SUMMARIZATION

4.1 Intro

In this chapter we shall explain why we need different kinds of summaries and how we build them. We analyze the algorithms and the tools used and how the resulting summaries get to be presented to the final user.

First of all, as we have already mentioned, we produce three kinds of summaries. The first one is a summary of a single web page. It is obvious that it is about the contents of a web page, from which the most important is extracted. The second kind of summary is that of a cluster. Contrary to the first one, except from extracting the most important content, it is necessary to extract content that is similar within the web pages of the specific cluster. Finally the third summary is also about a cluster. It is not exactly a summary; more like a few words describing it.

The following topics, using a similar formation, present a detailed analysis of all the above.

4.2 Single Page Summarization

The first thing to discuss is why there is a need for such summarization and the reason derives from the functionality of our application. After the user reaches, via the user interface, the level containing the results, once he finds a result he is interested in, he should be able to find out more about it. However, since the whole page content is rather unsuitable to display, due to its large size for such an application (bandwidth and space problems), the best way to inform the user, is extracting a summary out of it.

The nature of the application determines as well some characteristics of the summary. Specifically, it should be fast to extract, therefore our mechanism can't be really complex which leads us to an extract form of summary (representative sentences/phrases) instead of an abstract form of summary (a selected and processed summary around the basic subject). Since its purpose is to give the user an idea of what the page is about and not how it is related to his query, it should be query-independent too.

The implementation we followed is based on the title and location methods of Edmundson's proposal [16], modified to apply on web page content. Therefore, its main idea is to assign a score to each sentence of the web page and select those with the higher scores. The title method was to assign a higher score to sentences in title or headings. The location method was to assign higher score to sentences at the beginning of paragraphs or just under headings. Crossing over to web pages, this converts to assigning scores based on the html tags surrounding each sentence.

The source code of the implementation was written in Perl programming language. That was due to the convenience Perl provides when dealing with text and pattern matching. The steps we took in order to implement the algorithm an extract the summary are the following:

Step 1: Firstly we parse the webpage using the TreeBuilder Perl module. This reads the html code of the webpage and transforms it into an HTML syntax tree. The HTML syntax tree is used in order to find out under which tag each sentence belongs.

Step 2: Now we have to assign a score to the different html tags available. We assign different values from 100 down to 70 among important tags such as headers and paragraphs (the score goes to the first sentence. The rest sentences get the default score 0). We also give negative values to tags containing useless information regarding our summary (such as the <option> tag, which is used to enumerate items of a list). The final score of a sentence is the highest score of its surrounding tags.

Step 3: In order to define what we mean by saying ‘sentence’ we use the Text::Sentence Perl module. Still though, not every sentence is good enough to be in our summary. Really short sentences offer little information to the summary and very long sentences may take over the whole summary. So, we set a rule not allowing sentences of less than 30 characters or less than 3 words neither sentences with more than 250 characters to be in our summary.

Step 4: The final step is to define how long a summary will be. We set that limit to 400 characters as (by observation) it seems to be enough to produce a decent summary, taking under concern that a longer summary wouldn’t be appropriate for our application.

The whole process of extracting a webpage summary is triggered by a user request to get more details about that webpage. Thus, nothing needs to be done before that. All that is necessary is the user to supply which page he wants the summary for and the designed implementation returns the summary. All that is shown in figure 4.1 below.

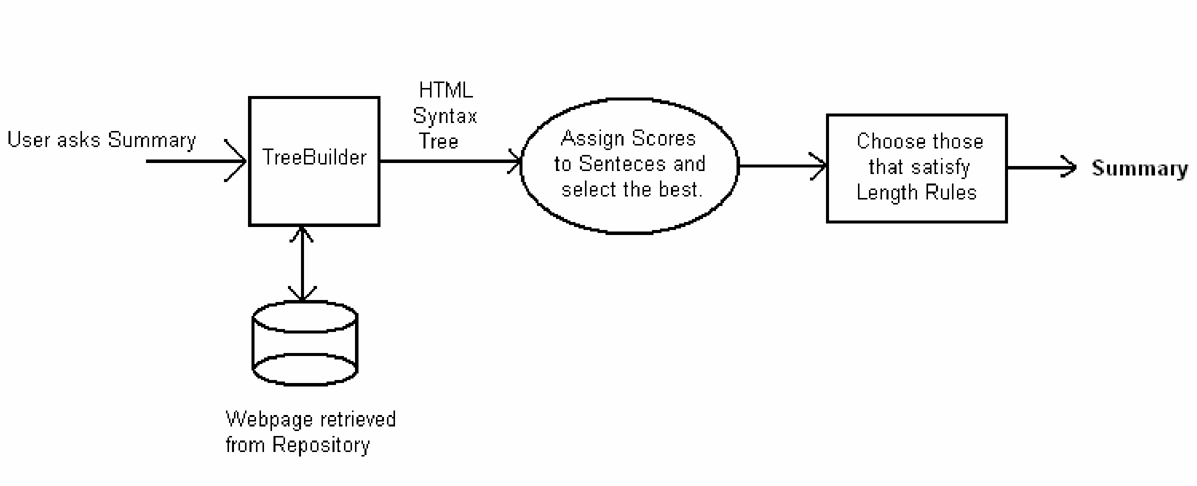

User asks Summary
TreeBuilder
HTML Syntax Tree
Assign Scores to Senteces and select the best.
Choose those that satisfy Length Rules
Summary
Webpage retrieved from Repository

Figure 4.1: Single Page Summarization

Judging the results of this summarization technique, we notice that indeed the summary is representative of the contents; this is expected since a) headings in html tend to withhold the most important information, and b) the first sentence of paragraphs is an introduction of what follows.

4.2 Cluster Summarization

The summary of a cluster is another important feature in our application. It can save the user time as he can have an idea of what the results of that cluster are, without having to descend hierarchy to reach them. Contrary to the previous case, now we don't have to deal with just one webpage but all the web pages of that cluster. This means that we have to follow a different approach. However, some issues remain the same. For example, we still have limitations in space and bandwidth so it must be short again.

To extract this multi-document summary, we took advantage of the MEAD summarization toolkit. As it will be explained later in this topic, MEAD can create a summary out of many documents using multiple algorithms. But, before using MEAD, some specific actions took place so that we transform the cluster web pages into an accepted by MEAD input form. The steps before using MEAD are presented below:

Step 1: The first thing to do is find out which pages belong to the specific cluster. Our clustering mechanism has already produced that, so we create a list of them and proceed to step 2.

Step 2: After we get the list of web pages from which we have to extract a summary, we extract textual data from them. This is done like that: Using Perl, we parse each html page and extract all text under title and heading tags. Then we proceed with the main text of the page which undergoes some further processing. We have to export not the full text but its sentences, so we once again define what a sentence is (in fact what a good summary sentence is – at least two words, starting with a capital and ending with a punctuation mark). At last, we have a list of all 'good' sentences of pages in the specified cluster.

Step 3: We have observed that phrases like "All rights reserved.", tend to appear in many web pages. The fact that such sentences have many appearances may reflect in our summary, so it would be wise to remove duplicate sentences in general. So, again using Perl, we remove all duplicate sentences from our list, and after that we have the final 'text' to summarize. We must notice that MEAD supports handling duplicate phrases but that feature is not fully developed; that's why we did this manually.

Step 4: The final step before using MEAD, is to transform the sentence list into a format that MEAD understands. This is the DocSent format presented later at the MEAD presentation. The transformation algorithm, since we already have the sentences extracted, was relatively simple and implemented once again in Perl.

Step 5: We use the created DocSent as input to MEAD and extract the summary. The respective command was:

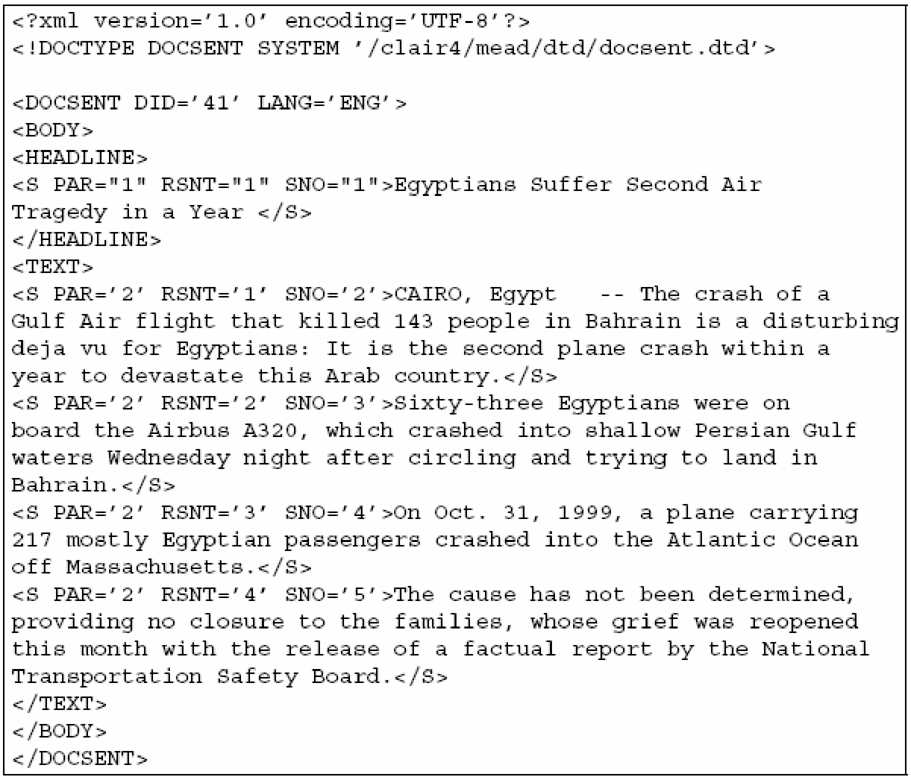
```
./mead.pl -s -a 4 -cluster_dir ../ClusterDir ClusterName;
```

The '-s' parameter means we want the output in sentences and the '-a 4' parameter means that the absolute number of sentences should be four. This number (four) is regarded as a good number of sentences in our application-oriented summary. Of course some more length limitations have been applied to the source code of MEAD. Specifically, we reckon that the total length of the summary should not by any chance be more than 800 characters.

4.2.1 MEAD Summarization Toolkit

Now, let's see a few more things of what MEAD is and how it extracts a multi-document summary. MEAD is a free toolkit for multilingual summarization and evaluation implementing numerous summarization algorithms such as position based, centroid, tf*idf, query based. It is written in Perl programming language, and uses additional external Perl modules.

It produces both lead based (selecting the first sentence of documents, then the second, etc.) and random summaries given a document or a cluster in DocSent format (figure 4.2).



```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE DOCSENT SYSTEM '/clair4/mead/dtd/docsent.dtd'>

<DOCSENT DID='41' LANG='ENG'>
<BODY>
<HEADLINE>
<S PAR="1" RSNT="1" SNO="1">Egyptians Suffer Second Air
Tragedy in a Year </S>
</HEADLINE>
<TEXT>
<S PAR='2' RSNT='1' SNO='2'>CAIRO, Egypt   -- The crash of a
Gulf Air flight that killed 143 people in Bahrain is a disturbing
deja vu for Egyptians: It is the second plane crash within a
year to devastate this Arab country.</S>
<S PAR='2' RSNT='2' SNO='3'>Sixty-three Egyptians were on
board the Airbus A320, which crashed into shallow Persian Gulf
waters Wednesday night after circling and trying to land in
Bahrain.</S>
<S PAR='2' RSNT='3' SNO='4'>On Oct. 31, 1999, a plane carrying
217 mostly Egyptian passengers crashed into the Atlantic Ocean
off Massachusetts.</S>
<S PAR='2' RSNT='4' SNO='5'>The cause has not been determined,
providing no closure to the families, whose grief was reopened
this month with the release of a factual report by the National
Transportation Safety Board.</S>
</TEXT>
</BODY>
</DOCSENT>
```

Figure 4.2: DocSent Format

Using the default parameters (for classifier, reranker) like we did, the scoring of a sentence **s** is a result (linear combination) of its features which are basically Position **P(s)**, Centroid **C(s)** and Length **L(s)**:

$$\mathrm{Score}(s) = w_P P(s) + w_C C(s) + w_L L(s)\,,$$

where **w** is the corresponding weight in the linear function. However additional sentence features can be used to calculate their score. The extract summary can be either absolute (exact number of sentences) or relative (percentage of the original text).

In general, the MEAD architecture is that of figure 4.3. Specifically there are the following steps:

a) Sentences become feature vectors.

b) Feature vectors become sentence scores.

c) Sentence score contribute to other sentence scores.

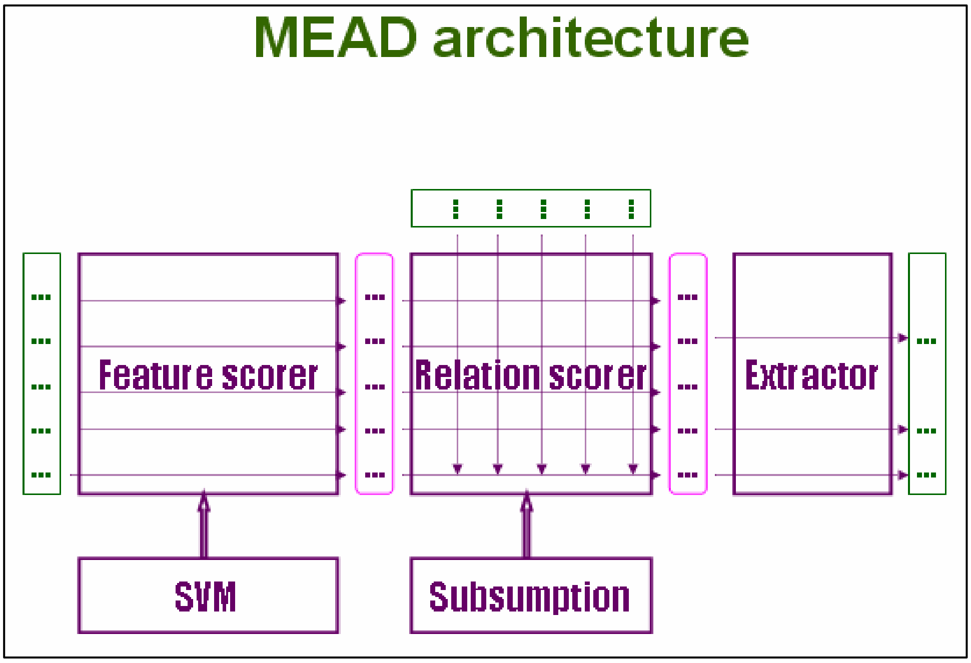


Figure 4.3: MEAD Architecture

It is important to note that MEAD could be used to extract the single webpage summary as well. However the algorithm we use is simpler, therefore faster and it has good results so we preferred it to MEAD.

Figure 4.4 shows the whole process of creating the multi-document summary for a cluster. As one can see, such an action is again triggered by the user's desire to get the cluster summary.

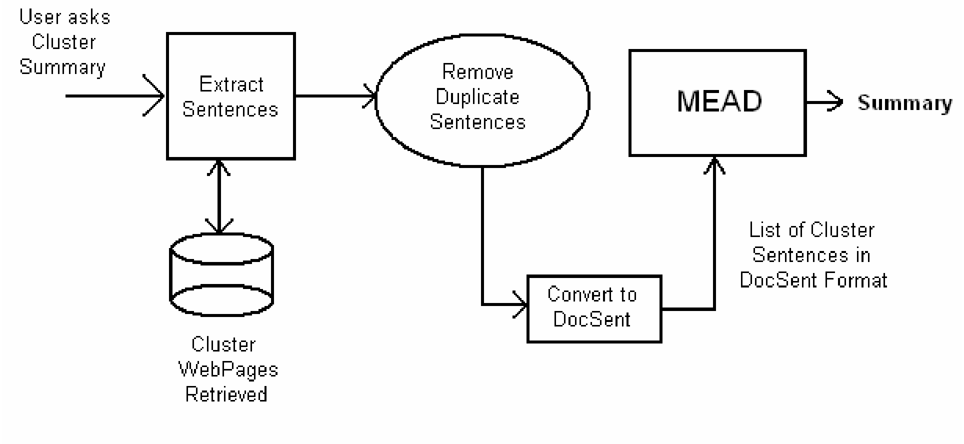

User asks Cluster Summary
Extract Sentences
Remove Duplicate Sentences
MEAD
Summary
Cluster WebPages Retrieved
Convert to DocSent
List of Cluster Sentences in DocSent Format

Figure 4.4: Cluster Summarization

An evaluation of the extracted summary of a cluster would show that it actually is representative of its results. However, the limited sentence number is negative factor in our attempt to be as precise as possible. This means that for large clusters, it fails to be as representative it is for shorter clusters.

4.3 Cluster Description

Contrary to the other summaries, the creation of a short description for every cluster does not derive from the user asking for it, but is necessary in order to present the available clusters to the user. This means it has to be extremely short and as precise as possible. Moreover it has to be fast to create since this may be done many times during a user's search.

Our approach was to use the sentences belonging to the cluster in order to find sequences of words and use the most frequent as the cluster description. Prior to that, some steps similar to those of cluster summarization were followed. A detailed analysis of those steps follows (with the final step being our main algorithm):

Step 1: Again we firstly have to find all pages belonging to the specified cluster and create a list.

Step 2: Like in the previous case all sentences were extracted from the cluster web pages using the same algorithm.

Step 3: Removing duplicate sentences is even more important than before since they contribute largely to the score of a word sequence. This may seem to be against our cause for the sentence redundancy is characteristic of the content, but in fact it is more important to find the same sequences in different sentences and not the replica of the same one. Thus, we use the previous algorithm so that we create a set of unique sentences.

Step 4: This step is about the use of our algorithm for finding sequences and their occurrences and assigning a score to them. The implementation is based on the N-gram theory and is written in Perl. An N-gram is a sequence of N words: $w_1w_2w_3w_4\ldots w_N$ and in our implementation N values we use vary between 7 down to 2. The algorithm we introduce is the following:

1. Create a list of stop-words. Those are words usually of great frequency but offering little or nothing information. Words from that list will be used in a cluster's description only if some certain rules get satisfied.
2. Calculate the occurrences of 7-grams, 6-grams and 5-grams appearing in the sentence list. Stop-words may appear in them.
3. Calculate the occurrences of trigrams and bigrams and unigrams appearing in sentence list as long as no stop-words are included. (Both in steps 2 and 3 we exclude numeric values and words 1 or 2 characters long).
4. We export the 7-gram, 6-gram or 5-gram with the most appearances, as long as it is appearing a fixed number of times (we selected 6, 7, 10 times accordingly) and we have not used it before.
5. If no higher N-gram is appropriate, we export a trigram but not that with the most appearances; a score is primarily calculated and that with the highest score extracted. The score is based on the appearance of its containing bigrams and unigrams. The rule giving that score is the following:

$$\text{score(3-gram)}=\frac{\text{count(3-gram)}*\sum_{\text{2-gram}\in\text{3-gram}}\text{count(2-gram)}}{\text{div}}$$

where **div** is given by the following rule:

$$\text{div}=\left(-2*\sum_{\text{2-gram}\in\text{3-gram}}\text{count(2-gram)}\right)+$$

$$+\left(\sum_{\text{1-gram}\in\text{3-gram}}\text{count(1-gram)}\right)$$

This rule by observation gives more accurate results, taking in mind that the more the occurrences of the containing bigrams are, while the containing unigrams are not that much, the higher the score for a trigram.

The following figure (4.5) is a schematic representation of the above procedure.

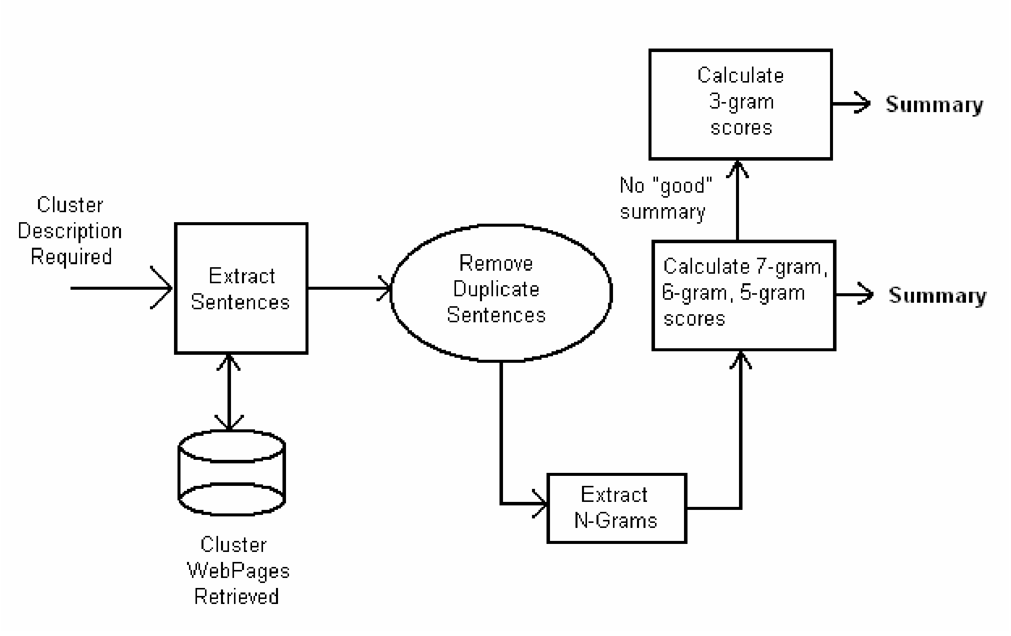

Calculate 3-gram scores
Summary
No "good" summary
Cluster Description Required
Extract Sentences
Remove Duplicate Sentences
Calculate 7-gram, 6-gram, 5-gram scores
Summary
Extract N-Grams
Cluster WebPages Retrieved

Figure 4.5: Cluster Description Extraction

The description of a cluster is being presented to the user when his parent cluster is displayed. Therefore, this procedure takes place just before showing the parent cluster. This means that prior to displaying the root cluster we have to extract description from all level 1 clusters. These descriptions are stored in our xml file; that finalizes its back-end produced content.

4.3 Where we stand

Now that we have all necessary back-end mechanisms, let's overview what we have achieved so far and what the next action should be.

First of all we have created an xml file which contains all necessary information in order to start displaying the results. This information regards both content and cluster structure. Moreover, we have all the procedures needed to extract more information from the back-end such as cluster and webpage summary – in order to be able to use them we just created the corresponding interfaces (executable programs that extract the desired summary) - that the user interface can call.

The clustering and summarization mechanism produce the desired results, so the next step is to implement the user interface for our application.

PART II

THE FRONT-END

In this part we deal with the front-end of our application; the user interface and the final application.

Chapter 5: In this chapter we demonstrate our user interface specifications, the handling of modalities and the design decisions we had to take.

Chapter 6: This chapter contains the implementation of the user interface and shows an example of use of the final application we have developed.

Chapter 7: Here we propose some future work that could be done both regarding our application and the surrounding fields of interest in general.

Chapter 5

UI SPECIFICATION

5.1 Intro

This part is about designing a user interface for the functionality offered by our back-end. The first step is to finalize all aspects regarding the final application; then to take the necessary design decisions concerning the multimodality issues of the user interface and finally implement it.

Regarding the final application, its purpose is more to act in means of showing the multimodal-driven platform that the back-end implements rather than to be a professional application. Thus, we decided that a web interface with a palm-like display, which however shows the synergies between modalities (keyboard-mouse input/text output and pseudo-speech input/output), will be enough. Before analyzing these modality issues, we should firstly demonstrate the final architecture of both the system and the user interface.

5.2 UI Architecture

Figure 1.1 displayed a general overview of the system architecture. A more detailed presentation can be seen in figure 5.1. As we can see it is more analytical of the system's modules and displays how the user interface positions between the user and the back-end. So, in order to understand what the user interface should offer, we have to define what actions the user can request and how they can be forwarded to the back-end. To do so we present the system specifications witch will allow us to partition the UI into functional blocks.

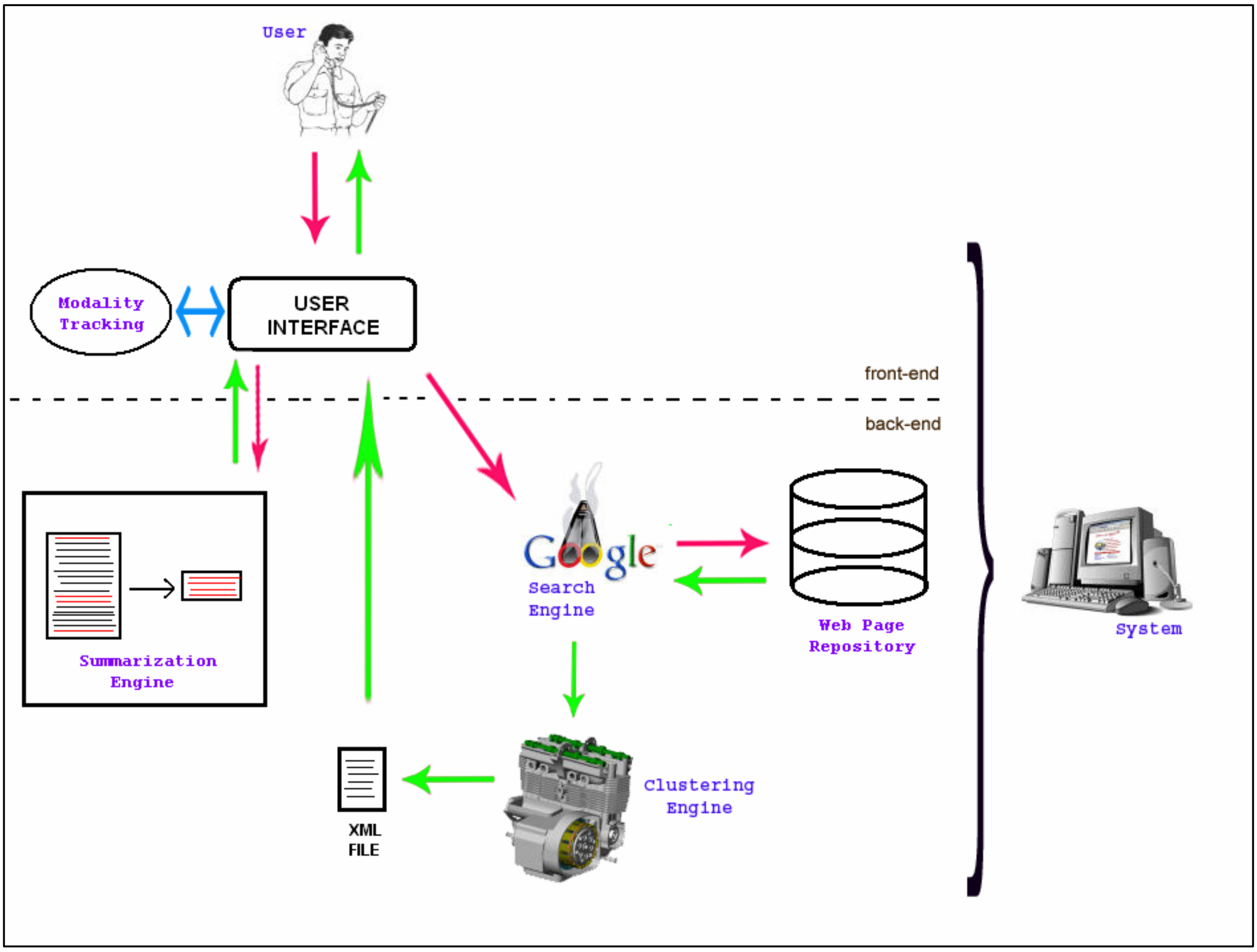


Figure 5.1: Complete System Architecture

5.2.1 System Specifications

System specifications are the following:

1. Connecting to the system: Users will connect to the system through the application and will be greeted by a prompt and a potential tutorial of how to use it.

2. Starting a Search: This is the basic functionality of the system. Users will submit a query and the system retrieves clustered results

3. Accessing clusters: Once the search engine has returned the query-related results, and they have been processed in the way we wanted them to be, the user will be told how many results have been found and the available options.

4. Navigation through the categories: To navigate through the clusters of each level, the user has only to say the name of the category (either select the appropriate category using his graphical interface). Moreover, the user is able to go to the parent category.

5. Requesting summary: For every cluster, the user will be able to request a summary of its results.

6. Getting top results: While in a non-result cluster (root or level 1, 2 cluster) the user can ask the system directly for the highest scoring results of the cluster

7. Display & Navigation through the results: Once the cluster results are retrieved, they are displayed in order of relevancy to the query. The system provides a title of each one to the users by which the user can ask for more details.

8. Asking for more information on a result: The user might find a result interesting and want to find out more about it. The system provides a more detailed summary on a result, should the user ask for it (and potentially additional information).

9. Additional Commands: The user may request to terminate current search and start another. The system has a main menu where the user selects the desired action.

10. Prompts and Help Messages: Different prompts will be given to the user according to the state hi is in. Help will be associated with the actions of the user. Repeated help requests will provide longer and more detailed help to the user.

5.2.2 UI Functional Blocks

The above system specification acts as the basis for the partitioning of the user interface into functional blocks. By saying functional blocks, we mean the different states of the application. The functional blocks will be used later to create the finite state machine of a user's flow in the system.

The blocks into which we separate the application are the following:

1. Initialization: The system greets the user and offers to him a tutorial of how to use it. The user can decide whether he wants the tutorial or not.
2. Tutorial: This is the block where the tutorial is displayed. The user can stop the tutorial whenever he wants. Both in this case and in case the tutorial ends, the user is lead to the main menu.
3. Main Menu: This block is the one that informs the user of the available functionality. At this moment the available functionality is limited to starting a search, listening to the tutorial and exiting the system. This block is the destination of every finished search.
4. Search: In this block the system asks the user for the search query. The system recognizes search string and initiates a query in the back-end. After the results are retrieved the user goes to the Cluster block.

5. Cluster: The system has retrieved the cluster's result (initial or lower level) and displays them to the user. In case of no results the user is informed about it and returns to main menu block. During this block a list of the cluster's sub-clusters is presented (using their description) and the user can select either one of them, or a cluster summary or a list of the top results for the cluster. Moreover he can ask the system to repeat available sub-clusters or he can select to return to a previous cluster. Selecting a sub-cluster or the parent cluster brings him back to a Cluster block.
6. Summary: The block that presents a cluster summary to the user following his request.
7. Results: It is like the Cluster block but for 3rd level clusters containing results. The list of result is presented and the user can get more details on a result by selecting the appropriate title.
8. Details: The block where a summary along with additional information about a result demonstrated.
9. Waiting Block: It's a block that has to do with waiting states within the application. Those states are produced when back-end processing is required. They inform the user of what is happening and offer him the ability to stop the running process
10. Help: This block is about providing context sensitive help. That is done after a user request for help, taking in mind what state he is in. When the help prompts are finished, the user returns to his previous state.
11. Confirmation: Block in which the system awaits for user confirmation of an action
12. Exit: User reaches this block when he chooses to exit the application. The system waves him with a goodbye prompt.

A further understanding of the UI functional blocks and how they interact with each other will come in chapter 5.3.1 where we display the activity diagrams for our application. Before that however we deal with the modalities of the final application and input/output of each block in general.

5.2.3 Modality Tracking

As we have mentioned in the beginning of this chapter, we will create a web interface for our application. It will follow a palm-like design, characteristic of the nature of application we designed. Specifically, we will use short prompts, and a limited Graphical User Interface focused on the available options for each state. Since it is a web interface, input will be keyboard/mouse actions over the html GUI and output will be an html page (text). However, in order to consider this application multimodal, it is necessary to import another type of input/output. That would be voice. We didn't use a voice recognition engine, so that is done by using pseudo-speech input prompts and by considering the text output as speech. To be more explanatory:

- We created a form for text input that follows the structure of natural language. It shall act as an equivalent of using voice. When the user uses that form, what he said (typed) is parsed to extract a command that the system recognizes. This form can be used only when the system allows "speech" input. That is when a)the GUI is not in use and b)the state the user is in allows speech input (every state does, but some of them not before finishing a prompt)

- In order to emulate speech output, we don't accept user input until specific prompts are spoken (time delay). Moreover we follow the techniques applying to voice applications; short prompts, constant disposal of the available actions to the user.

Figure 5.2 shows the modality tracking procedure used.

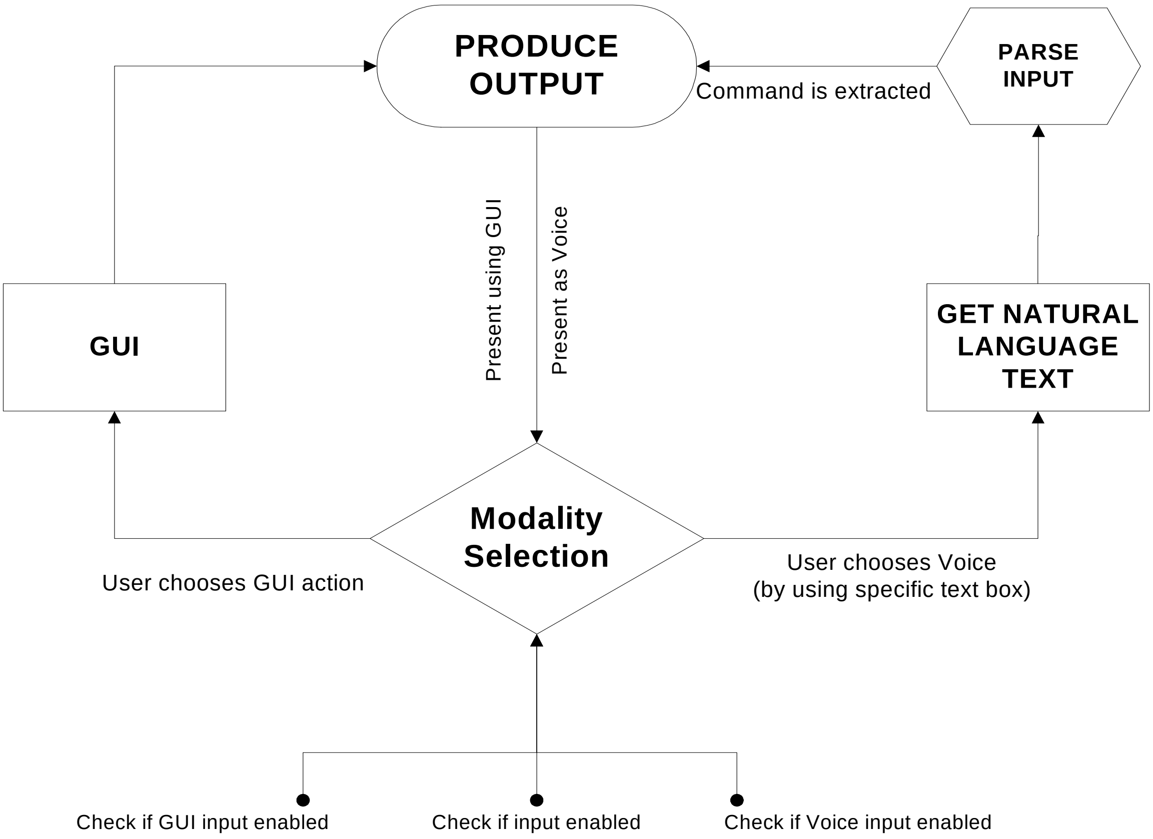


Figure 5.2: Modality Tracking

In case the user uses the GUI, the next action of the system is obvious; it's what the user selected. However, when using natural language input, the system is unable to understand the user unless some parsing is primarily done. Thus, we use various input grammars to parse user input. The grammars and their inputs we use are the following:

1. System created input grammars: Based on the cluster descriptions and result titles.
2. Search input grammars: Grammar based on words one can find in the web pages that constitute our repository.
3. Confirmation grammars: "that's correct", "that's right", "yes",
4. Rejection grammars: “no”, “not really”, “negative”.

5. Summary grammars: "summary", "details", "more information"
6. Exit input grammars: "exit", "cancel", "stop", “skip”
7. Help inputs: "help", "tutorial".
8. Cluster related input grammars: "repeat", "top results"
9. Back grammars: “back”, “return”
10. Jump grammars: “go to the initial cluster”, “go to third result”

Apparently, the inputs shown above are just a subset of the whole grammars. The grammars we created support a much larger number of requests whether they are imperative or polite.

We should finally note that while the user is at one state, the available pseudo-voice input he can submit comes to total correspondence with the action provided by the GUI.

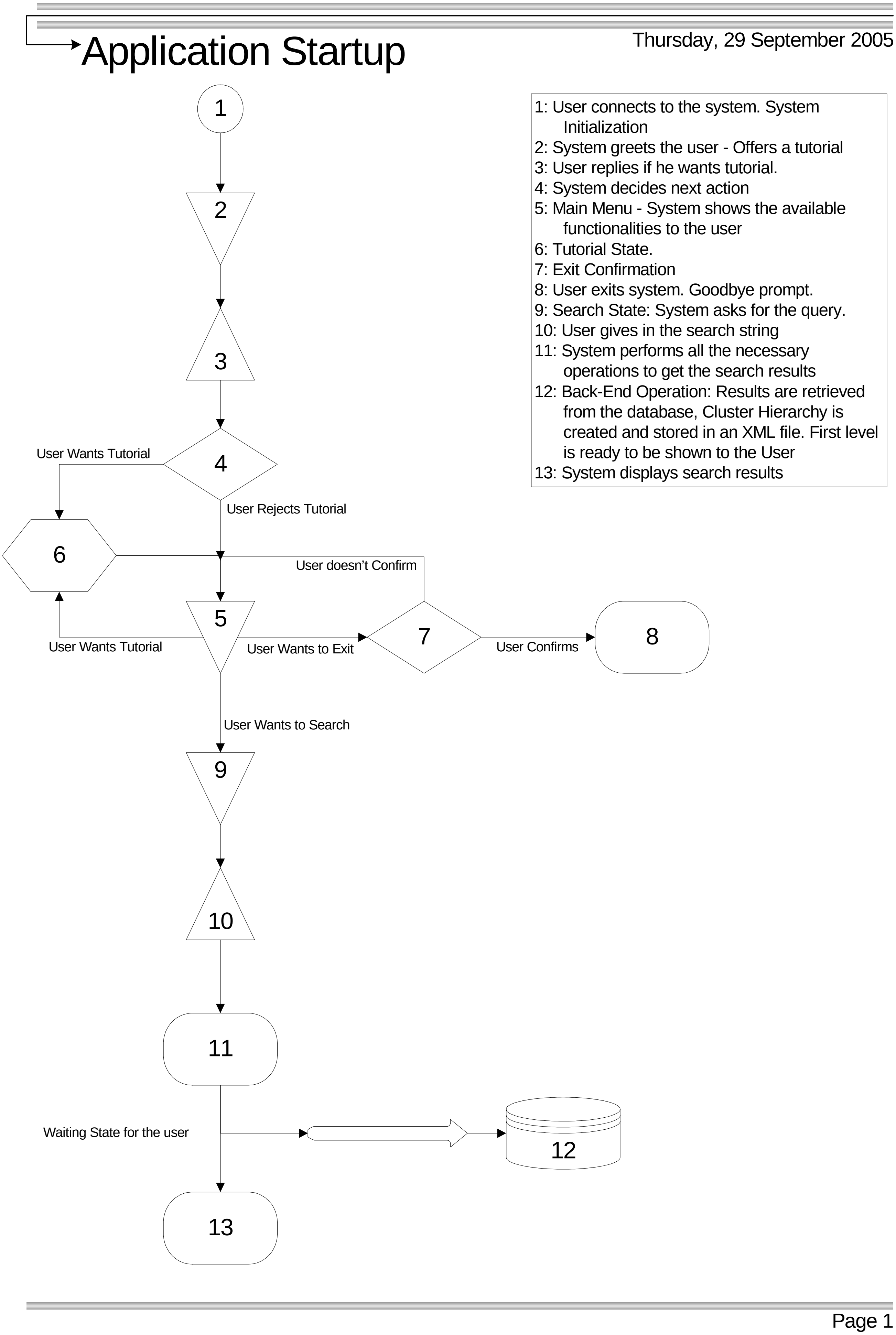
5.3 UI in Use

Before presenting use cases of the final application, we shortly demonstrate what kinds of input are accepted in each state and show the activity diagrams of the Finite State Machine that simulates a user’s route in the system.

5.3.1 Activity Diagrams

Here follows a description of the relation between inputs and states. The definition of this relation is the step prior to creating the FSM. Starting from the initial state, the user can either accept or reject the tutorial. If he accepts, the tutorial is offered but he can skip anytime. The next state, the main menu, accepts input regarding search, help or exit procedures. In case of ‘exit’ input, a confirmation is necessary. However the user can’t reply until the system prompt is over. That is done due to the importance of his answer. When he selects to search, he can either enter a search string or cancel. As soon as the results return to him, being in a cluster block, he can either ask for a repeat, for a summary, for top results or to navigate to a previous cluster. No voice input, until the basic options are explained, allowed. Result level clusters have the same options, instead that of a summary. Moreover the selection of the result leads the user to a result's details instead of a cluster. Top details state is similar to that of a result level cluster. Finally summary states are similar to detail states; user can only return back or cancel the search.

The following activity diagrams show the user flow, according to his inputs and the connection with the back-end.



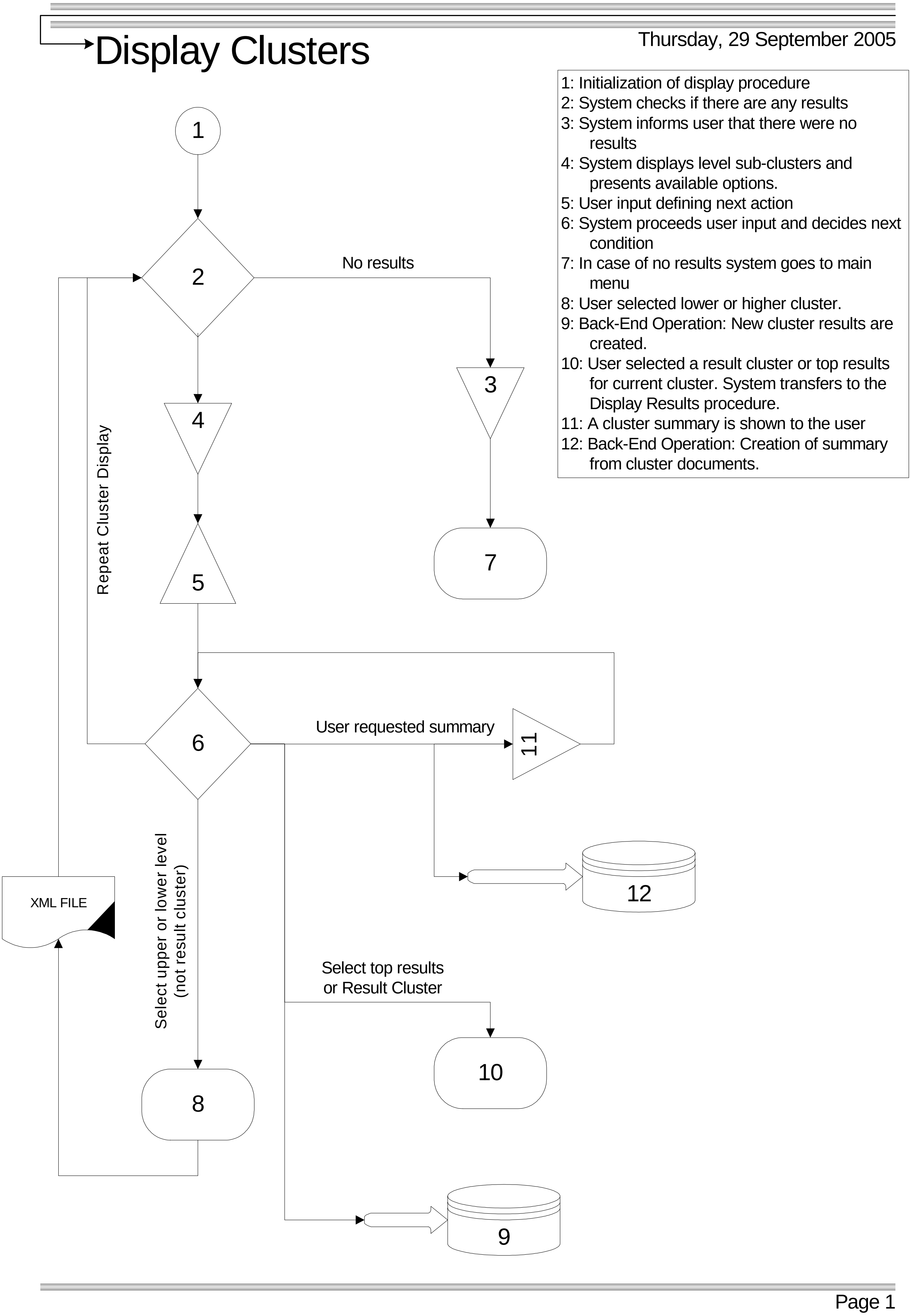
Application Startup

Thursday, 29 September 2005

1: User connects to the system. System Initialization
2: System greets the user - Offers a tutorial
3: User replies if he wants tutorial.
4: System decides next action
5: Main Menu - System shows the available functionalities to the user
6: Tutorial State.
7: Exit Confirmation
8: User exits system. Goodbye prompt.
9: Search State: System asks for the query.
10: User gives in the search string
11: System performs all the necessary operations to get the search results
12: Back-End Operation: Results are retrieved from the database, Cluster Hierarchy is created and stored in an XML file. First level is ready to be shown to the User
13: System displays search results

Page 1

Figure 5.3: Application Start-up Activity Diagram



Display Clusters

Thursday, 29 September 2005

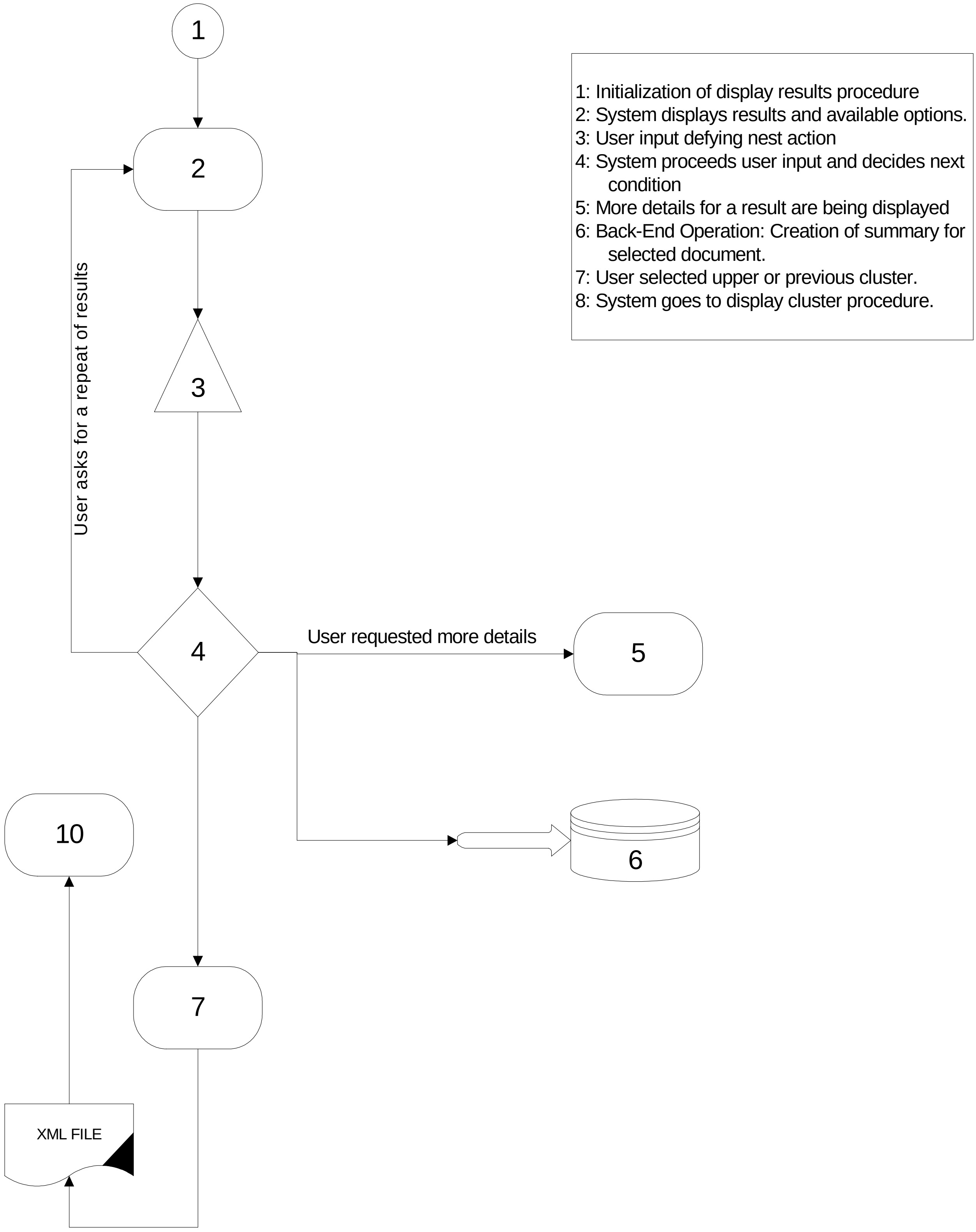
1: Initialization of display procedure
2: System checks if there are any results
3: System informs user that there were no results
4: System displays level sub-clusters and presents available options.
5: User input defining next action
6: System proceeds user input and decides next condition
7: In case of no results system goes to main menu
8: User selected lower or higher cluster.
9: Back-End Operation: New cluster results are created.
10: User selected a result cluster or top results for current cluster. System transfers to the Display Results procedure.
11: A cluster summary is shown to the user
12: Back-End Operation: Creation of summary from cluster documents.

Page 1

Figure 5.4: Cluster Display Activity Diagram

Display Results

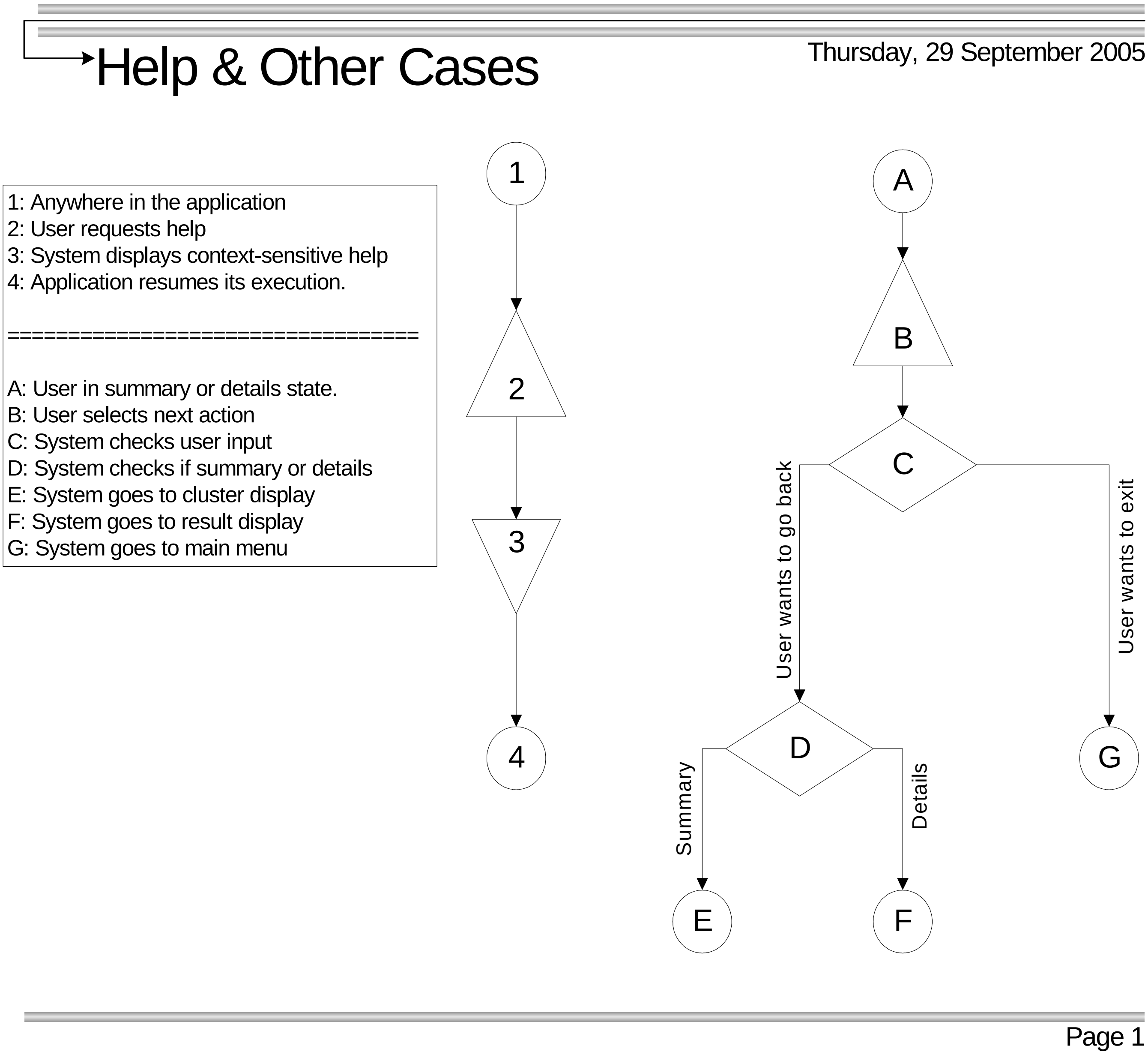
Thursday, 29 September 2005


1
2
3
User asks for a repeat of results
4
User requested more details
5
6
10
7
XML FILE

1: Initialization of display results procedure
2: System displays results and available options.
3: User input defying nest action
4: System proceeds user input and decides next condition
5: More details for a result are being displayed
6: Back-End Operation: Creation of summary for selected document.
7: User selected upper or previous cluster.
8: System goes to display cluster procedure.

Page 1

Figure 5.5: Result Cluster Display Activity Diagram



Help & Other Cases

Thursday, 29 September 2005

1: Anywhere in the application
2: User requests help
3: System displays context-sensitive help
4: Application resumes its execution.

================================

A: User in summary or details state.
B: User selects next action
C: System checks user input
D: System checks if summary or details
E: System goes to cluster display
F: System goes to result display
G: System goes to main menu

Page 1

Figure 5.6: More Application Activity Diagram

5.3.2 Use cases Revisited

Now that we have built the back-end and made the necessary decisions for the user interface we can go on with its implementation. Before doing that, let's see how one of the previously presented use cases transforms taking in mind what we've done so far. (Additional interaction added in order to show more features of the application)

(S) Welcome to the voice search engine. Would you like to listen to a tutorial of how to use the system?

(U) Yes I do.

(S) The Voice Search engine is… [Tutorial continues]

(S) You are at the main menu. You can either start a new search or exit the system. You also can listen to the system tutorial. What would you like to do?

(U) Help!

(S) In order to start a new search say SEARCH. To listen to a tutorial say TUTORIAL. To exit, just say EXIT.

(U) I would like to start a new search please.

(S) Please enter the search string for your query.

(U) "Chania Holidays"

(S) Please wait until the system retrieves all relevant documents. This process may take a while.

(S) There are 123 results in 3 sub-clusters. In order to select a sub-cluster say its name. By saying SUMMARY a summary of the cluster is presented, while TOP RESULTS returns a list of the highest scoring results, Available sub-clusters are: 1) Prefecture of Chania, 2) Greek Islands, 3) Winter Holidays.

(U) Give me a summary.

(S) Here is a short summary of the selected cluster: Chania is located in Crete, an island in the southern Greece. Chania is well known as a tourist resort both for the summer and winter because of its mild climate. This is the end of the summary. You can either go BACK or CANCEL this search.

(U) User selects to go back using the GUI

(S) There are 123 results in 3 sub-clusters. In order to select a sub-cluster say its name. By saying SUMMARY a summary of the cluster is presented, while TOP RESULTS returns a list of the highest scoring results, Available sub-clusters are: 1) Prefecture of Chania, 2) Greek Islands, 3) Winter Holidays.

(U) "Chania"

(S) You have selected "Chania". There are 81 results in 5 sub-clusters. In order to select a sub-cluster say its name. By saying SUMMARY a summary of the cluster is presented, while TOP RESULTS returns a list of the highest scoring results, Available sub-clusters are: 1) City of Chania, 2) Travel Guides of Chania, 3) Platanias, 4) Paleochora, 5) Sfakia

(U) "Platanias"

(S) You have selected "Platanias". There are 12 results in 4 sub-clusters. In order to select a sub-cluster say its name. By saying SUMMARY a summary of the cluster is presented, while TOP RESULTS returns a list of the highest scoring results, Available sub-clusters are: 1) Accessibility, 2) Hotels in Platanias, 3) Restaurants in Platanias, 4) Nightlife in Platanias.

(U)"Hotels"

(S) You have selected "Hotels". There are 3 websites that matched your query. Ask for "More Details" in order to get more information on a specific result. Available results are:

(S) 1) Apladas Hotel. A first rate hotel complex, offering a special experience for your summer holidays. 'www.apladas.gr'

(U) User selects more details for the given result using the GUI

(S) Here follows a summary of selected result. Brand new and imposing APLADAS BEACH is located directly on the sandy beach of Platanias village in Chania-Crete. APLADAS BEACH with its relaxing atmosphere and the high standard facilities focuses on making summer holidays an unforgettable experience. This is the end of the summary. You can either go BACK or CANCEL this search.

(U) Quit.

(S) You are at the main menu. You can either start a new search or exit the system. You also can listen to the system tutorial. What would you like to do?

(U) Exit please.

(S) You have selected to exit the application. Is that correct?

(U) Yes that's right.

(S) Thank you for using the voice search engine.

Figure 5.7: Example Use Case 2

Chapter 6

UI IMPLEMENTATION

6.1 Intro

This chapter starts with the procedure followed in order to implement the user interface presented in the previous chapter. After it is implemented, the application will have reached its final stage. All that remains is to test it and evaluate it. Later in this chapter a real use of the application is presented and the results are analyzed.

6.2 Building the UI

We mentioned before that we shall build a web user interface. In fact, we will design the user interface using a combination of Java servlets and JSPs (Java Server Pages). Servlets and JSPs are part of the Enterprise Java and a very commonly used tool to create web services. Concerning our case, we use them in order to accommodate a possible future transformation into a standalone application, since the only thing to change will be to change the ways of getting the input and displaying the output.

The basic issues that the user interface implementation has to support are the following three:

1. The creation of a user interface (GUI and pseudo-voice UI) according to the FSM discussed in the previous chapter, with the required input/output capabilities.
2. The ability tie change information with the back-end and activate its mechanisms.
3. The ability to parse pseudo-voice input according to the given grammar and understand what the user meant.

Starting with the first issue, we implement the UI with a series of servlets and JSPs that correspond to the FSM's states. Specifically we use JSP pages to implement the graphical and pseudo-voice UI of states that offer some information to the user (e.g. prompts, replies, etc.). In general, they contain simple html code for the display of the output and the creation of GUI forms. Servlets are used to handle transitions between states and to communicate with the back-end. A transition from one state to the other is therefore a transition from one JSP to another. Which will be this second JSP, depends on the user input.

However user input is not enough to make some transitions. For example when having a 'back' command, we have to know what the previous state was. Thus, using Java programming language, we implement a Record class that supports the handling of additional information during a user's rambling through the application. In specific a Record contains information regarding the state one is in, the level of the cluster hierarchy he has reached, the cluster he is in and the initial query. A Record is constantly updated whenever some of its containing information changes.

In order to use this Record object, it is attached to a Session Object. The Session Object is offered by the Servlet API and its purpose is to withhold information of a user connected to the application server. It initializes every time the user connects to our application, and we deactivate it when he disconnects. A Session supports the attachment of other objects onto it, which it carries until deactivated. So, by accessing a user's Session we are able to handle the information his Record holds.

Example uses of the Record object one can find while navigating through the clusters (we know what cluster he came from and what level he's reached) or asking for help (in that case the help system shows context sensitive help which derives from the state the user is in as shown by his Record).

So far we have explained how the display and the transitions are implemented. Regarding the drafting of information from the back-end we haven't said much. This is done using the XML file that has been created and the additional interfaces that our back-end provides.

Taking things sequentially, until the user sends a search query, there is no communication with the back-end. When our front-end gets that query, it uses one of the available interfaces of the back-end to start a search. To be more specific, this interface is in fact a batch executable that handles the search engine, the clustering and summarization engines and finally exports the desired XML file. In order to run that executable we use the specific API that Java provides. After the query-related XML file is exported from the back-end, our front-end communicates only with it, except for the case of getting additional summaries (then it uses an interface to communicate just with the summarization engine).

In order to communicate with the XML file we use the *dDom4j* API. *Dom4j* uses the Document Object Model (DOM) to read and write to an XML file. Both DOM and the *dom4j* API are presented a little later. For the moment, let's see how we use the information from that file.

The first thing the user interface does is to display the sub-clusters of the root cluster. All the necessary information (which are the sub-clusters and their description) is already in the XML file. So, using the *dom4j* API we read that and present them to the user via the corresponding JSP. When a sub-cluster is selected, a similar procedure is done. However, there is some information missing in this case. This is the descriptions of the new cluster's sun-clusters. To extract them, we use summarization engine interfaces. Apart from using the extracted description just to display it to the user, we also store it in the XML file; that way we can have it available for future use without having to include the summarization engine in that process.

A similar procedure is followed when the user requests summary for a cluster or for a result. Since this information is not available until needed, we use the rest summarization interfaces to extract it. Again this information is stored in the xml file.

Summing up, it is clear that the XML file storing all this information makes it easier for us to handle from the UI. Moreover, it acts like a cache, reducing time when a summary is already extracted. In general, the algorithm determining the user interface – back-end interaction can be seen in figure 6.1

1) Activate back-end procedures that create the XML file for a given query
2) Use XML file to retrieve information regarding the display of Root Cluster.
3) For every request for a sub-cluster, a summary or a result's details, check the XML file.
    - If all information available, present it to the user
    - If not available, extract it from the back-end and a)store it to the XML file and b) present it to the user

Figure 6.1: Information Extraction Algorithm

6.2.1 DOM and dom4j

The DOM (Document Object Model) is a platform- and language-neutral interface which allows programs and scripts to dynamically access and update the content, structure and style of (HTML and XML) documents. Along with SAX (Simple API for XML - An event-driven, serial-access mechanism for accessing XML documents), they are currently the two most popular APIs for manipulating XML documents. Using DOM, the document can be further processed and the results of that processing can be incorporated back into the presented page. Behind the interface the document is represented with an object-oriented model. By supporting the DOM API, a program not only allows its data to be manipulated by other routines, but does so in a way that allows those manipulations to be reused with other DOMs, or to take advantage of solutions already written for those DOMs. In general, the DOM API provides a standardized, versatile view of a document's contents.

Dom4j is an easy to use, open source library for working with XML, XPath and XSLT on the Java platform using the Java Collections Framework and with full support for DOM, SAX and JAXP.

In our implementation, *dom4j* provided the DOM platform; all the necessary APIs to access, read, traverse and write an XML file. Specifically, it created a structure of all elements and their attributes available in the XML file. In order to access the value of an element's attribute, we only had to provide the attribute's XPath (unique address in XML file). Any manipulations made to these values could be stored by transforming the altered structure again into an XML file.

Moreover *dom4j* supports the creation of new elements. This is helpful when we extract a summary and we want to store it in the XML file.

6.2.2 Parsing the Input

The final issue we have to take in mind following our implementation is that of parsing the pseudo-speech input. As we have mentioned that is done using specific grammars. The parsing is based on a simple Perl program that matches the input with a given grammar file. This parser returns what kind of request it was (as long as it was a valid request) and potentially some additionally information provided in the input (for example when the user wants to go back N steps, we have to know the value of N). According to the kind of request that was, the servlet handling the input decides the next action.

The first step to set up the parsing module is to assign a grammar for every state. A BNF grammar format is used to describe our grammars. What our Perl script does, is to find if the user input is accepted by the state grammar and under which rule. The rule is what determines the command. This is done using pattern matching techniques. Moreover, the understanding of the value of certain inputs is also supported. To do so, we use the pattern matching wildcard and store the recognized part of the input, so that we can use it later. This happens for example when the user enters the query, or when he selects to go N steps back and we have to know the value of N.

In most states grammars are static. That means that we know from the beginning the accepted inputs. However, when clusters are displayed the grammar of those states must contain a cluster's description or a result's title. This is how we deal with it: First of all we create the static part of the grammar. While the descriptions or titles are displayed, they are added to our grammar. This, beyond others, means that the user can't select something that hasn't been displayed (or 'said' for the speech-output consideration).

6.3 VoiceSearch in Action

Putting together all modules, the application (VoiceSearch) is ready to run. In order to do that, we hosted the JSPs and the compiled servlets under a Tomcat web server. By giving the specified address of the initial state JSP to any browser, we were able to run and test the created application. The figures of Appendix A display the application in a usage scenario.

6.3.1 User Studies

Here we present the usage scenario for the figures in Appendix A that show a use case of the final application system. This usage scenario goes like this:

1. The user enters the system and accepts to listen to the tutorial (1)
2. The system displays the tutorial (2)
3. The system goes to the main menu. User selects to start a new search (3)
4. The system asks the user for the search query. User enters “charisteas euro 2004 final” (4)
5. The system asks the user to wait for the results (5)
6. As soon as the results are retrieved the system displays the root cluster. User selects one of the available sub-clusters (6)
7. The new cluster is being displayed. User asks for summary (7)
8. The cluster’s summary is presented to the user (8)
9. The user returns to previous cluster and now he asks for its top results (9).
10. The top results for the cluster are being displayed (10)
11. The selected result's details are displayed to the user (11)
12. The user follows the necessary actions to return to the root level cluster
13. at the root level cluster, the user selects a different sub-cluster (12)
14. Selecting one of the available sub-clusters each time, the user descends to a result level cluster.
15. The cluster's results are being displayed. User selects the only result (13)
16. The selected result's details are displayed to the user (14)
17. User selects to cancel the search.
18. User goes to main menu. He selects to exit the application.
19. The system needs a confirmation (15)
20. After the user confirms, the system goodbyes the user and clears his session (16).

6.4 Conclusions

We tested the application we created using multiple scenarios and search queries. Concerning the flow of the user into the system, we can see that at any given time, the available options are obvious to him by the user interface. Testing the help and error systems (the error system is in fact a prompt informing the user that the system couldn't understand him and what he should do) we are assured of their good operation. Moreover the navigation through states is exactly like we designed it. The FSM transitions are absolutely followed.

Regarding the results of the queries we tested, it became obvious that we had good results only for queries related somehow with the content of the stored websites (sports and academic issues). Irrelevant queries produced few results and sometimes without clear connection between them. However that was expected at some point, due to the relatively small in size repository.

We could say that the clustering mechanism worked quite well. Specifically we could realize that by testing "university of Cambridge" related queries which produced clusters based on the various departments of it. Summaries of web pages and clusters, as mentioned before, also produce decent results. The only summary of less quality was that of the cluster description. That was when the results, although similar in words were not similar in word sequences (which is the key in our algorithm extracting cluster descriptions).

The user interface presents the results in n acceptable manner, while the modality selection doesn't produce any problems (due to the way it was implemented).

Summing up, the application we have design fulfills its initial purpose of using features like clustering and summarization with a positive outcome. It also provides a simple user interface with multimodal support and based on a platform specifically focused low bandwidth/space devices.

Chapter 7

FUTURE WORK

7.1 Intro

Finally we shall discuss how we can extend the application we have implemented and the future work on the fields of interest. As we have mentioned before, those fields are under heavy research so at any time new features develop and can become a part of a multimodal search engine. We will focus on both back-end and front-end extensions, so that we improve the basis of the application as ell as the human-system interaction.

7.2 Extending VoiceSearch

The first thing we have to look upon is the further improvement of the algorithms we use. This has to be done in a way that no heavy processing (that will increase the response time) is required. Specifically for clustering, the use of links between pages (that is already used by the HITS based search engine) could make clusters even more distinct. Apart from that, cluster merging techniques can be applied so that clusters similar to each other with few results can be merged together in a more general cluster.

Regarding the summarization techniques, a good addition would be to use query-based summarization for cluster summaries. In particular, the cluster's summary will contain information based on the cluster's description. The cluster's description algorithm can be improved as well. This can be done by extending the list of stop-words, and using only nouns (which usually withhold the most important information) to create the description. Finally, link analysis could be again of help to create a summary [17].

Many improvements can be applied to the front-end. Probably the closest addition shall be the creation of a real Voice Interface. That can be easily done (since the modality tracking algorithm has been introduced) by attaching a Speech Recognition system to the input and a TTS (Text to Speech) system to the output of the application. Also, DTMF support could be easily added (especially for cluster and result selection – they are already displayed as a numbered list). GUI integration could also take place.

However, the most important extension that can be done is the extension of the functionality offered. Possible additional functionalities could be the following:

- **Personalisation of service**

  The users will be able to create their own accounts, so that the system understands them. They will be able to define some personal settings (like the way the results are displayed, if a cluster summary is required for every cluster without asking for it, what would be the systems initial state, etc). This would mean a signup and login procedure prior to using the system.

- **Search Filtering**

This is about filtering the already retrieved results using a new search query. For example, after an initial query Q_1 returns N results, the user will be able to search between those N results using a new query Q_2.

- **Additional Search Features**

The user can search the system for web pages containing an exact phrase. Also the query could contain binary operators that define the user's needs.

- **Taking advantage of the Summarization Engine**

The user could ask directly for a web page summary that already exists in our repository. The only thing to provide would be its internet address. Another possible functionality would be to get the summary of the initial cluster produced by a query. This is similar to an electronic encyclopaedia's functionality (for example, a summary created of the results of a "Diego Maradona" query, would return who was Diego Maradona).

- **Taking advantage of the Clustering Engine**

The user could ask directly for topics related to the content of a web page. Again, the only thing to provide would be its internet address.

- **Multilingual Support**

  Currently only the English language is supported. Using the appropriate stemmers, this application could apply to other languages as well.

Regardless to everything else, an evaluation of the application is necessary to take place. This means that it should be tested by many users so that its good operation is confirmed. Such tests can extract useful information (and therefore any weak points that need improving will come to the surface) about the following:

1) How accurate is the clustering.
2) How accurate is the summary of a cluster or a result.
3) What changes should take place on the UI.
4) To what point is multimodality better than using just one type of input.

Finally, the fact that our design is based on low bandwidth/space devices makes it easy to transform the application for any kind of such devices or even for classic computers.

References

[1] A. Arasu et al, "Searching the Web", ACM Trans. on Internet Technology, Vol. 1, No. 1, Aug. 2001, pp. 2-4

[2] L. Page, S. Brinn, R. Motwanni, T. Winograd, "The PageRank citation ranking: Bringing order to the Web", Computer Systems Laboratory, Stanford University, Stanford CA, 1998

[3] S. Brinn, L. Page, "The Anatomy of a Large-Scale Hyper-Textual Web Search Engine", Computer Science Department, Stanford University, Stanford, CA 94305, 1998

[4] G. Salton et al., "The SMART System-Experiments in Automatic Document Processing", Prentice-Hall, Englewood Cliffs, N.J., 1971

[5] E. Voutsakis, "Image Retrieval on the World Wide Web", Thesis Proposal, Technical University of Crete, 2004

[6] Monica R Henzinger, "Hyperlink Analysis for the Web", IEEE Internet Computing vol. 5, 2001

[7] J. Kleinberg, "Authoritative sources in a hyperlink environment". J. ACM 46, 6 (Nov.), 1999

[8] I. Dhillon, J. Fan and Y. Guan, "Efficient Clustering of very large Document Collections", Kluwer Academic Publishers, 2001

[9] http://www.cs.utexas.edu/users/yguan/datamining/spkmeans.html

[10] Dawid Weiss, "A Clustering Interface for Web Search Results in Polish and English", Thesis Proposal, Poznan University of Technology, 2001

[11] R. Osdinn, I. Ounis and R. White, "Using Hierarchical Clustering and Summarization approaches for Web Retrieval", TREC 2002 Interactive Track, Glasgow

[12] O. Zamir and O. Etzioni, "Grouper: A Dynamic Clustering Interface to Web Search Results", Computer Networks Journal, Amsterdam, Netherlands, 1999

[13] Iwona Białynicka-Birula, "Clustering Web Search Results", Department of Informatics of University of Pisa

[14] D. Cutting, D. Karger, J. Pedersen, J. Tukey, "Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections",

Proceedings of the Fifteenth Annual International {ACM} {SIGIR} Conference on Research and Development in Information Retrieval, 1992

[15] Andrew W. Moore, “K-means and Hierarchical Clustering”, Computer Science Department, Carnegie Mellon University, 2001

[16] H. P. Edmundson, “New Methods in Automatic Extracting Full text”, Source Journal of the ACM (JACM) Volume 16, Issue 2, April 1969

[17] J. Delort, M. Rifky, B. Bouchon-Meunier, “Enhanced Web Document Summarization Using Hyperlinks”, University of Paris, 2003

[18] J Goldstein, “Automatic Text Summarization of multiple Documents”, Thesis Proposal, Carnegie Mellon University, 1999

[19] H. Saggion, R. Gaizauskas, “Multi-Document Summarization by Cluster/Profile Relevance and Redundancy Removal” University of Sheffield, 2004

[20] www.summarization.com/mead

[21] Dragomir R. Radev and Simone Teufel and Horacio Saggion and Wai Lam and John Blitzer and Hong Qi and Arda Celebi and Danyu Liu and Elliott Drabek, “Evaluation challenges in large-scale multi-document summarization: the MEAD project”, Proceedings of ACL 2003

[22] http://www.cre.canon.co.uk/_texts/csweb.html

[23] http://larbin.sourcefourge.net/

[24] T. V. Raman, “User Interface Principles For Multimodal Interaction”, http://www.almaden.ibm.com/cs/people/tvraman/chi-2003/mmi-position.html

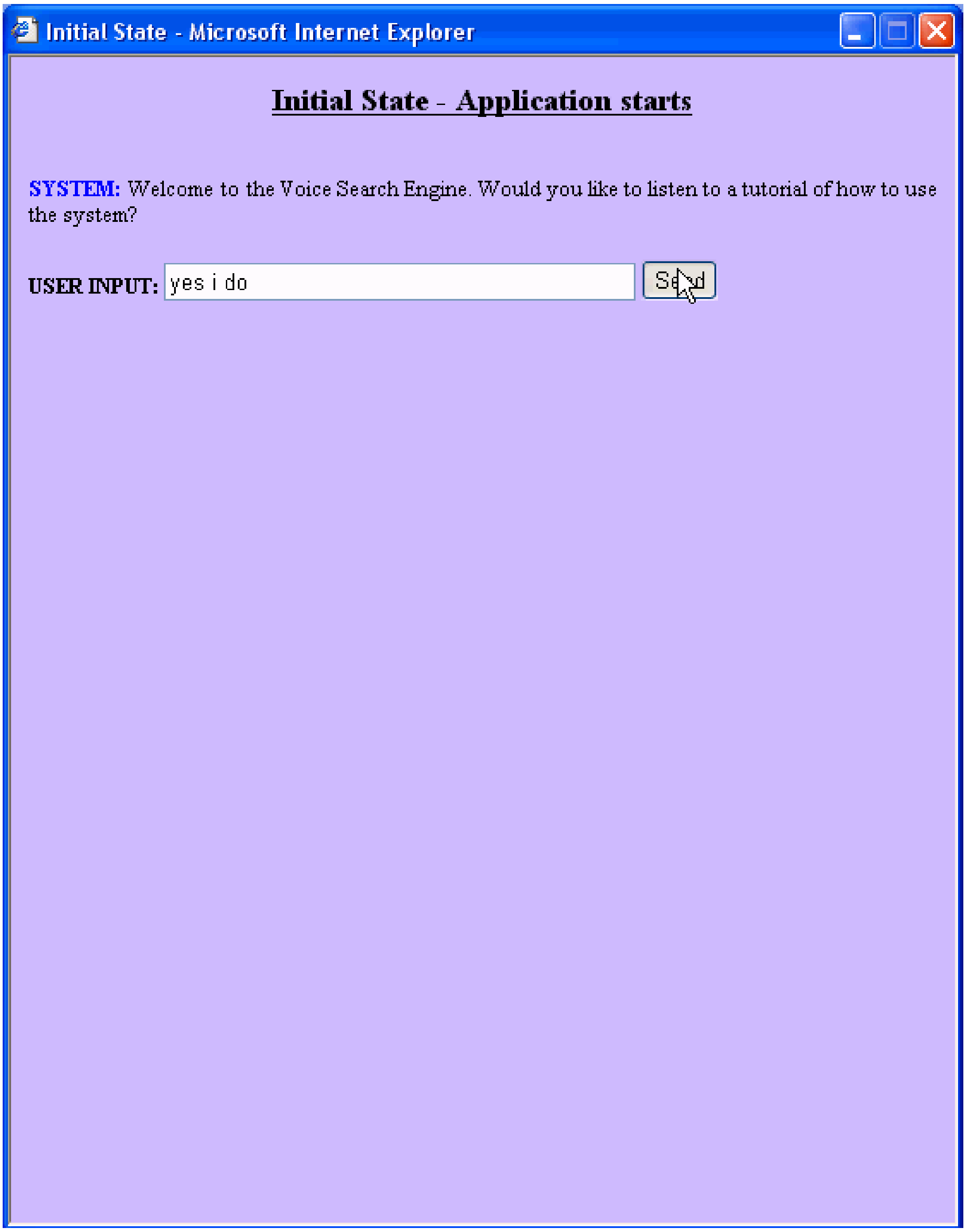
[25] Sharon Oviatt, Phil Cohen, Lizhong Wu, John Vergo, T. J. Watson, Lisbeth Duncan, Bernhard Suhm, Josh Bers, Thomas Holzman, Terry Winograd, James Landay, Jim Larson & David Ferro, “Designing the User Interface for Multimodal Speech and Pen-based Gesture Applications: State-of-the-Art Systems and Future Research Directions”, 2000

[26] Anoop K. Sinha and James A. Landay, “Embarking on Multimodal Interface Design”, Group for User Interface Research, Computer Science Department, University of California at Berkeley,

Appendix A

Here is the application output following the scenario presented in 6.3.1.

1

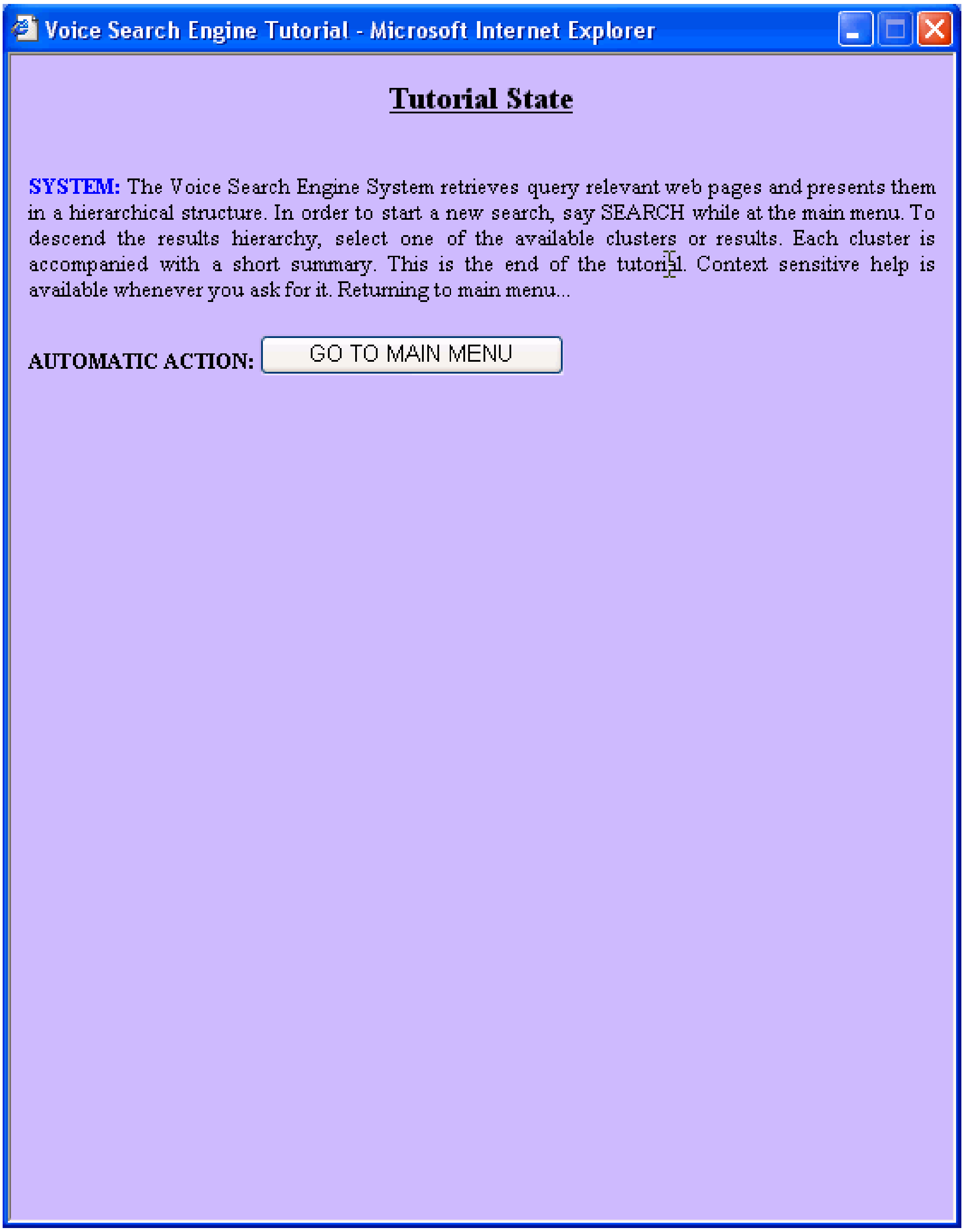


Initial State - Application starts

SYSTEM: Welcome to the Voice Search Engine. Would you like to listen to a tutorial of how to use the system?

USER INPUT: yes i do

2

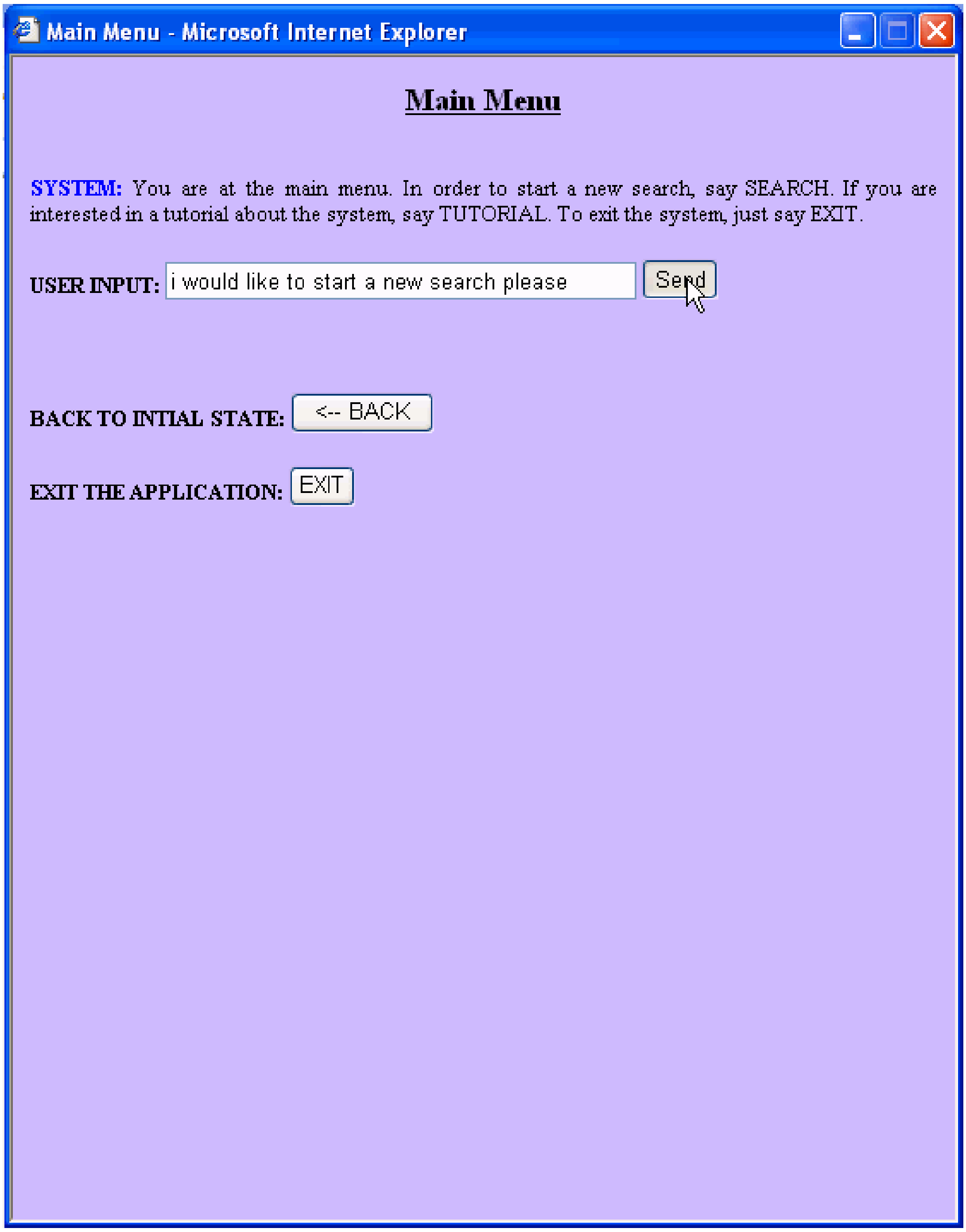


Tutorial State

SYSTEM: The Voice Search Engine System retrieves query relevant web pages and presents them in a hierarchical structure. In order to start a new search, say SEARCH while at the main menu. To descend the results hierarchy, select one of the available clusters or results. Each cluster is accompanied with a short summary. This is the end of the tutorial. Context sensitive help is available whenever you ask for it. Returning to main menu...

AUTOMATIC ACTION: GO TO MAIN MENU

3



Main Menu

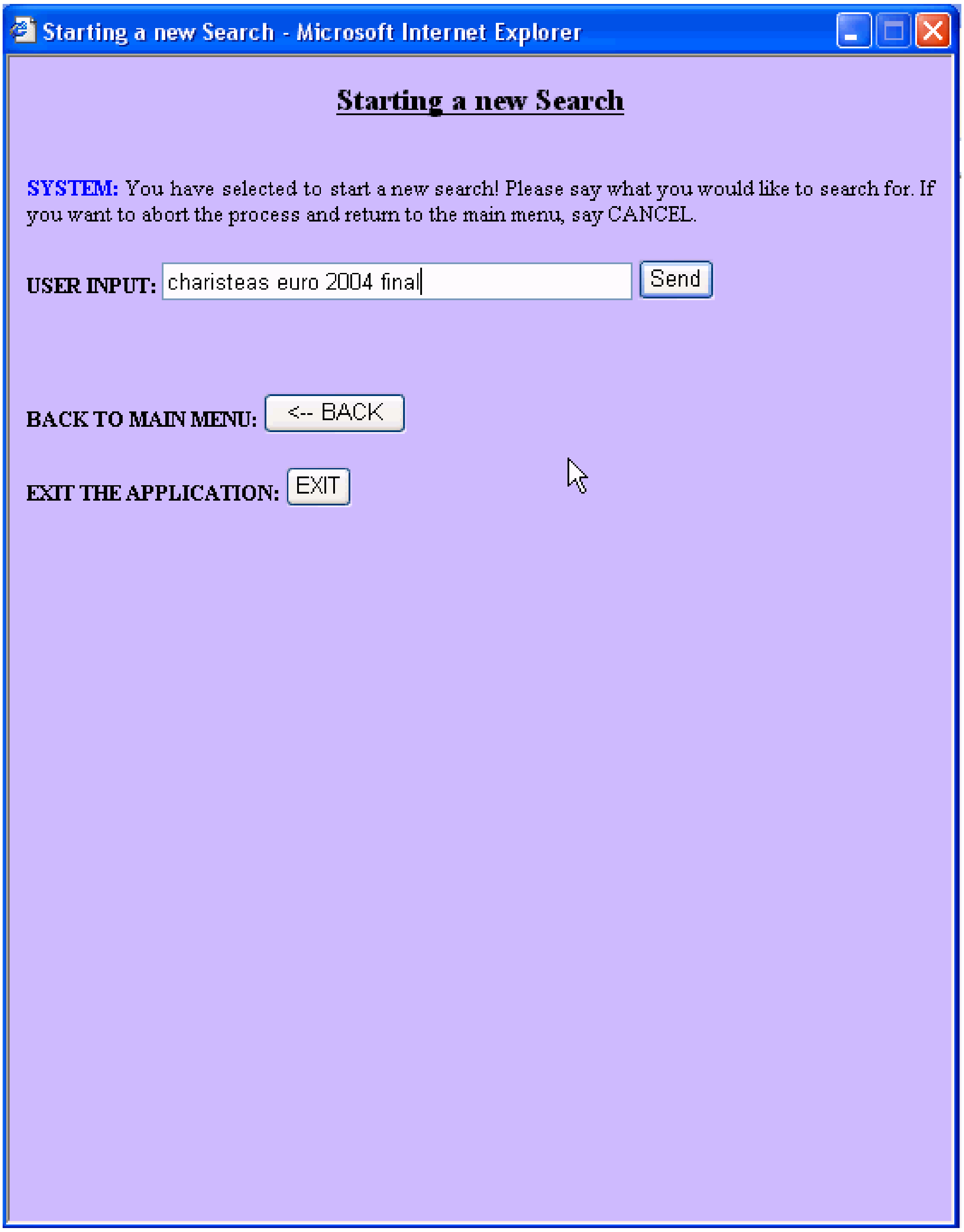
SYSTEM: You are at the main menu. In order to start a new search, say SEARCH. If you are interested in a tutorial about the system, say TUTORIAL. To exit the system, just say EXIT.

USER INPUT: i would like to start a new search please

BACK TO INTIAL STATE:

EXIT THE APPLICATION:

4



Starting a new Search

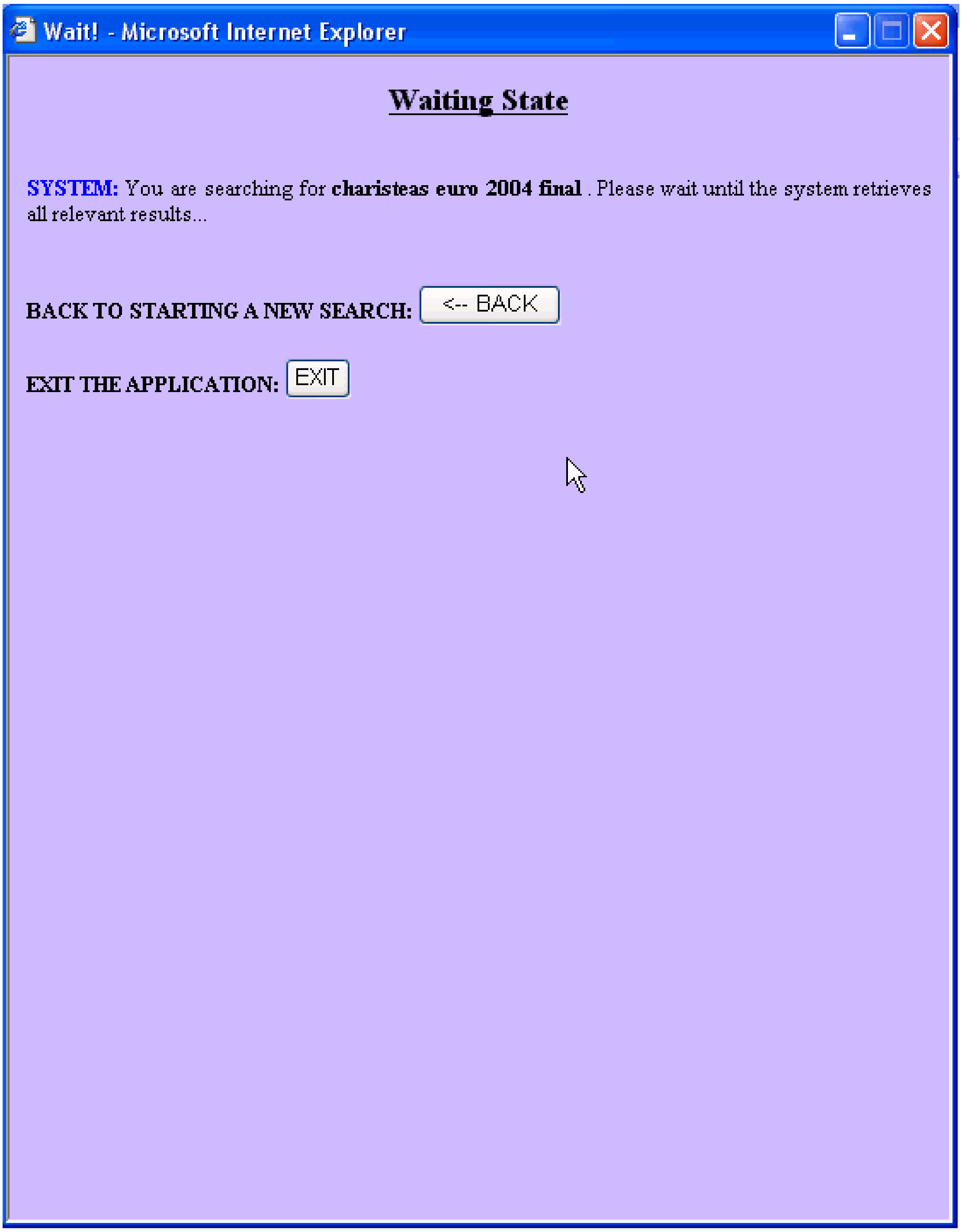
SYSTEM: You have selected to start a new search! Please say what you would like to search for. If you want to abort the process and return to the main menu, say CANCEL.

USER INPUT: charisteas euro 2004 final

BACK TO MAIN MENU:

EXIT THE APPLICATION:

5



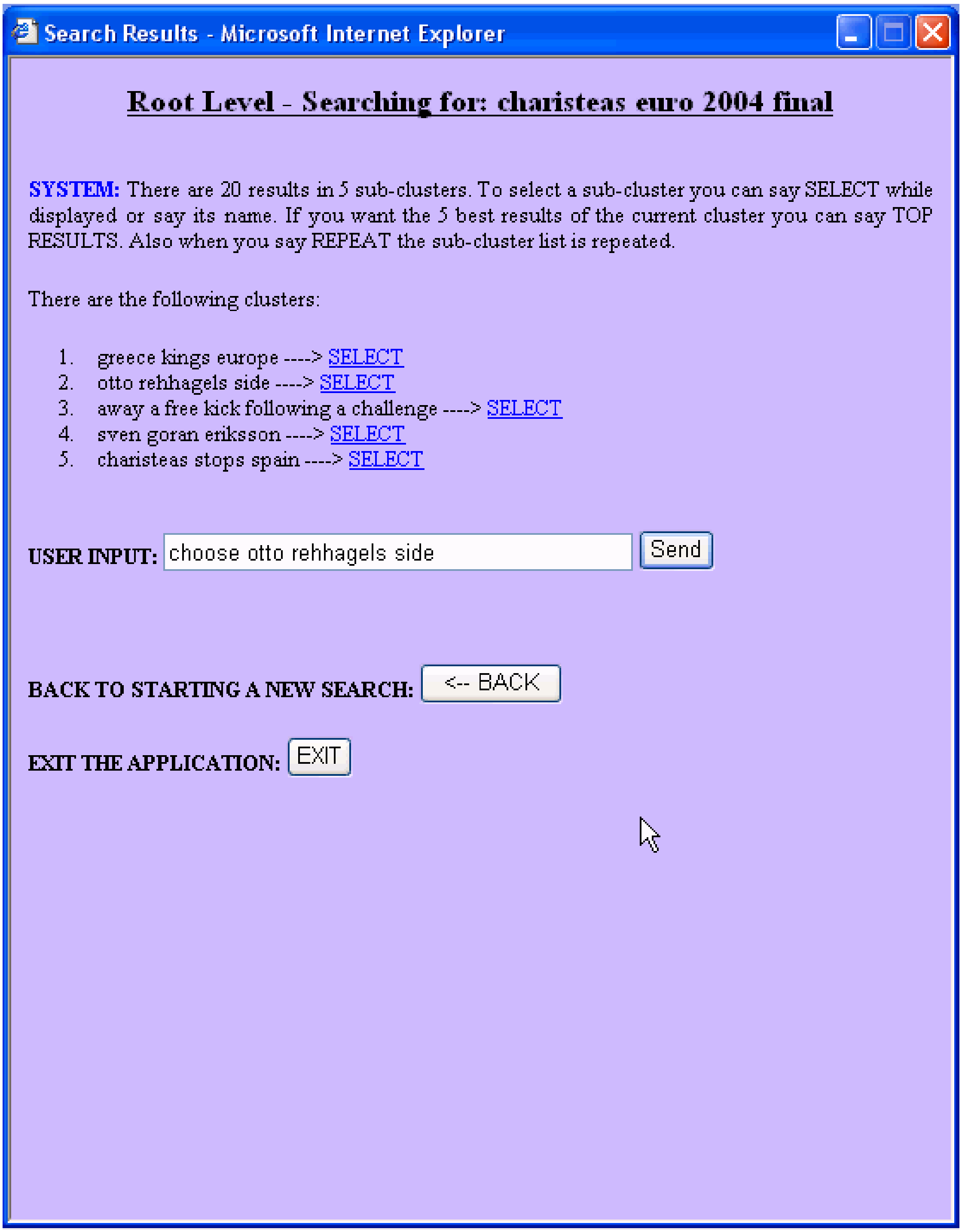
Waiting State

SYSTEM: You are searching for **charisteas euro 2004 final** . Please wait until the system retrieves all relevant results...

BACK TO STARTING A NEW SEARCH: <-- BACK

EXIT THE APPLICATION: EXIT

6



Root Level - Searching for: charisteas euro 2004 final

SYSTEM: There are 20 results in 5 sub-clusters. To select a sub-cluster you can say SELECT while displayed or say its name. If you want the 5 best results of the current cluster you can say TOP RESULTS. Also when you say REPEAT the sub-cluster list is repeated.

There are the following clusters:

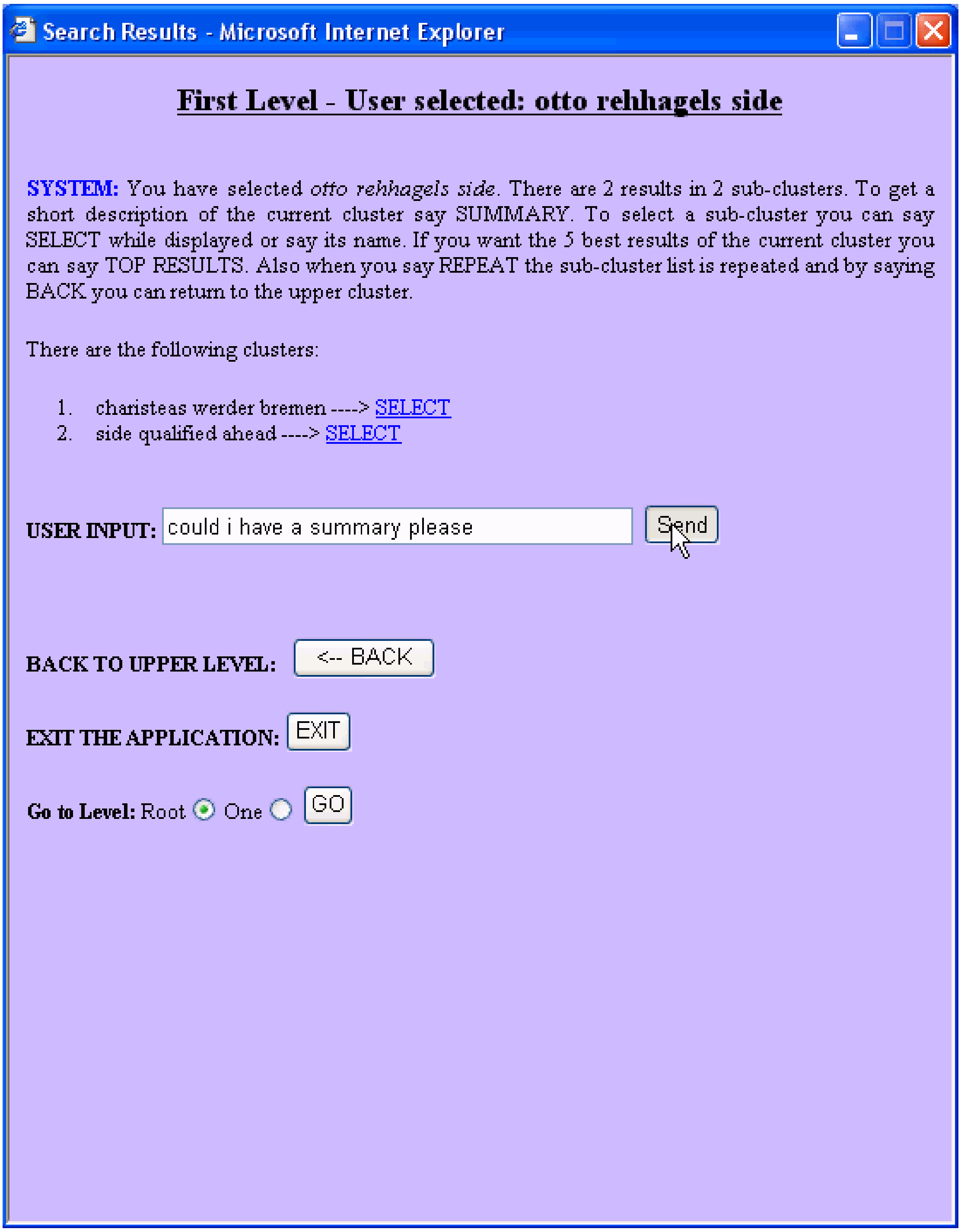
1. greece kings europe ----> SELECT
2. otto rehhagels side ----> SELECT
3. away a free kick following a challenge ----> SELECT
4. sven goran eriksson ----> SELECT
5. charisteas stops spain ----> SELECT

USER INPUT: choose otto rehhagels side

BACK TO STARTING A NEW SEARCH:

EXIT THE APPLICATION:

7



First Level - User selected: otto rehhagels side

SYSTEM: You have selected *otto rehhagels side*. There are 2 results in 2 sub-clusters. To get a short description of the current cluster say SUMMARY. To select a sub-cluster you can say SELECT while displayed or say its name. If you want the 5 best results of the current cluster you can say TOP RESULTS. Also when you say REPEAT the sub-cluster list is repeated and by saying BACK you can return to the upper cluster.

There are the following clusters:

1. charisteas werder bremen ----> SELECT
2. side qualified ahead ----> SELECT

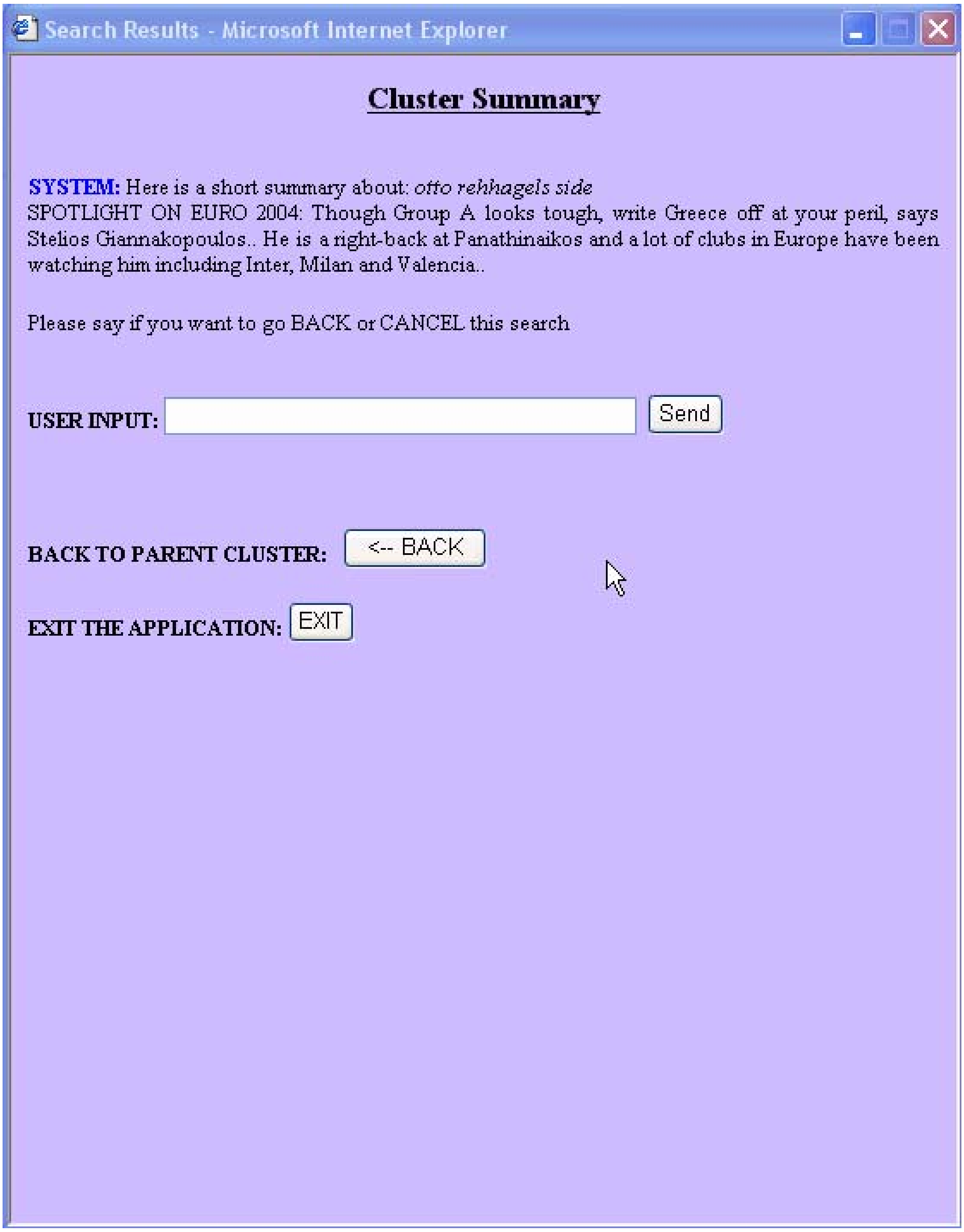
USER INPUT: could i have a summary please

BACK TO UPPER LEVEL:

EXIT THE APPLICATION:

Go to Level: Root ● One ○

8



Cluster Summary

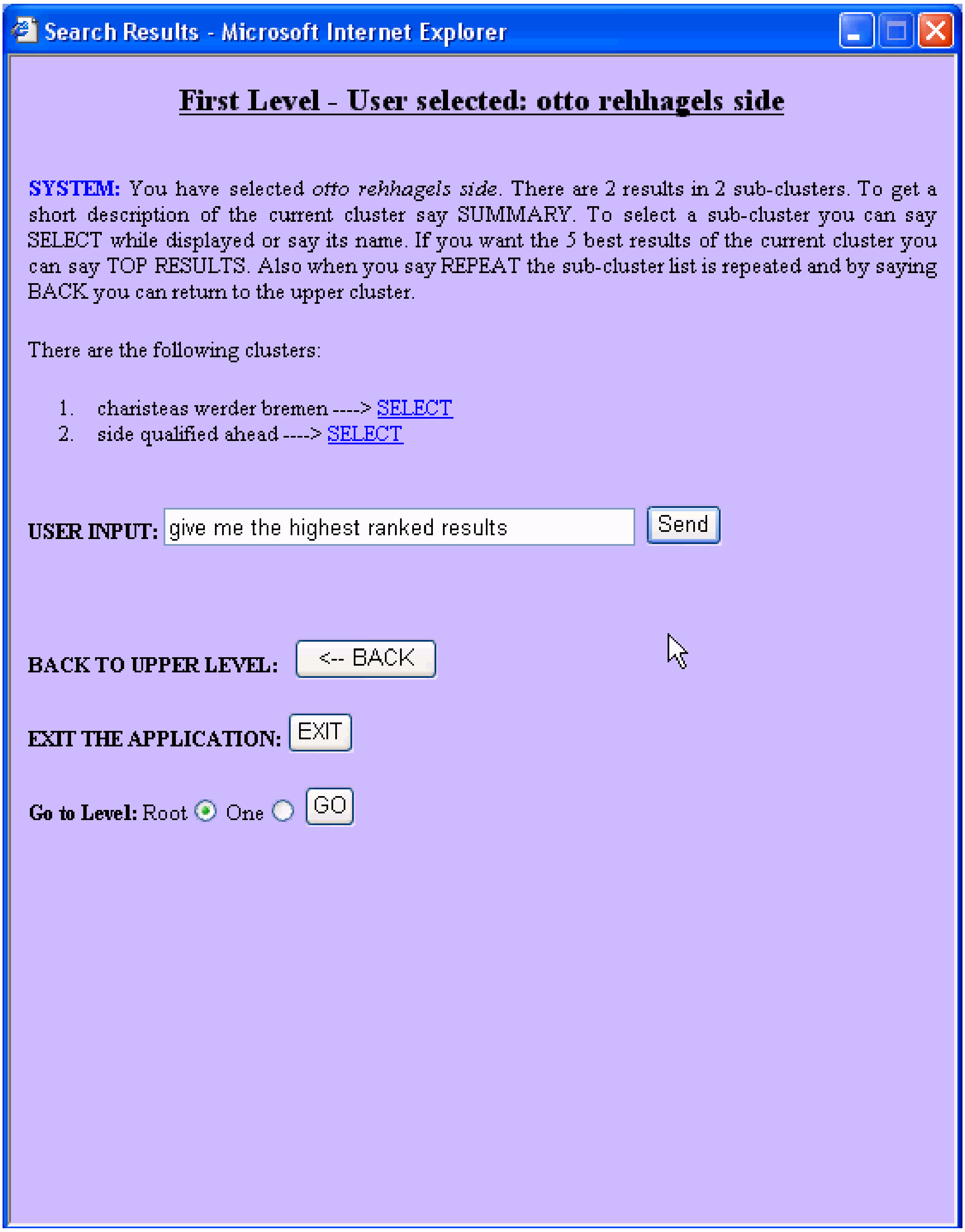
SYSTEM: Here is a short summary about: *otto rehhagels side*
SPOTLIGHT ON EURO 2004: Though Group A looks tough, write Greece off at your peril, says Stelios Giannakopoulos.. He is a right-back at Panathinaikos and a lot of clubs in Europe have been watching him including Inter, Milan and Valencia..

Please say if you want to go BACK or CANCEL this search

USER INPUT:

BACK TO PARENT CLUSTER:

EXIT THE APPLICATION:



First Level - User selected: otto rehhagels side

SYSTEM: You have selected *otto rehhagels side*. There are 2 results in 2 sub-clusters. To get a short description of the current cluster say SUMMARY. To select a sub-cluster you can say SELECT while displayed or say its name. If you want the 5 best results of the current cluster you can say TOP RESULTS. Also when you say REPEAT the sub-cluster list is repeated and by saying BACK you can return to the upper cluster.

There are the following clusters:

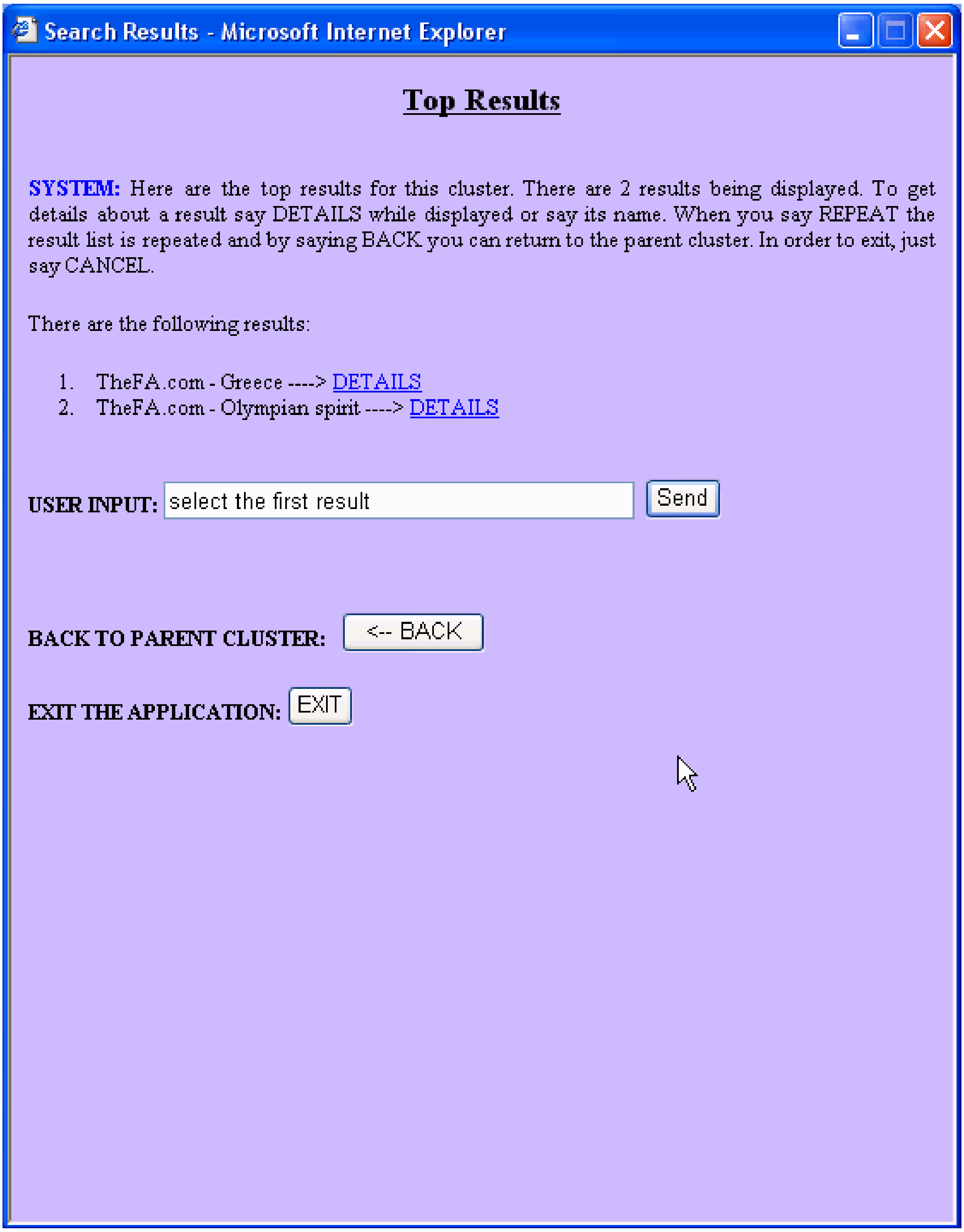
1. charisteas werder bremen ----> SELECT
2. side qualified ahead ----> SELECT

USER INPUT: give me the highest ranked results Send

BACK TO UPPER LEVEL: <-- BACK

EXIT THE APPLICATION: EXIT

Go to Level: Root ● One ○ GO



Top Results

SYSTEM: Here are the top results for this cluster. There are 2 results being displayed. To get details about a result say DETAILS while displayed or say its name. When you say REPEAT the result list is repeated and by saying BACK you can return to the parent cluster. In order to exit, just say CANCEL.

There are the following results:

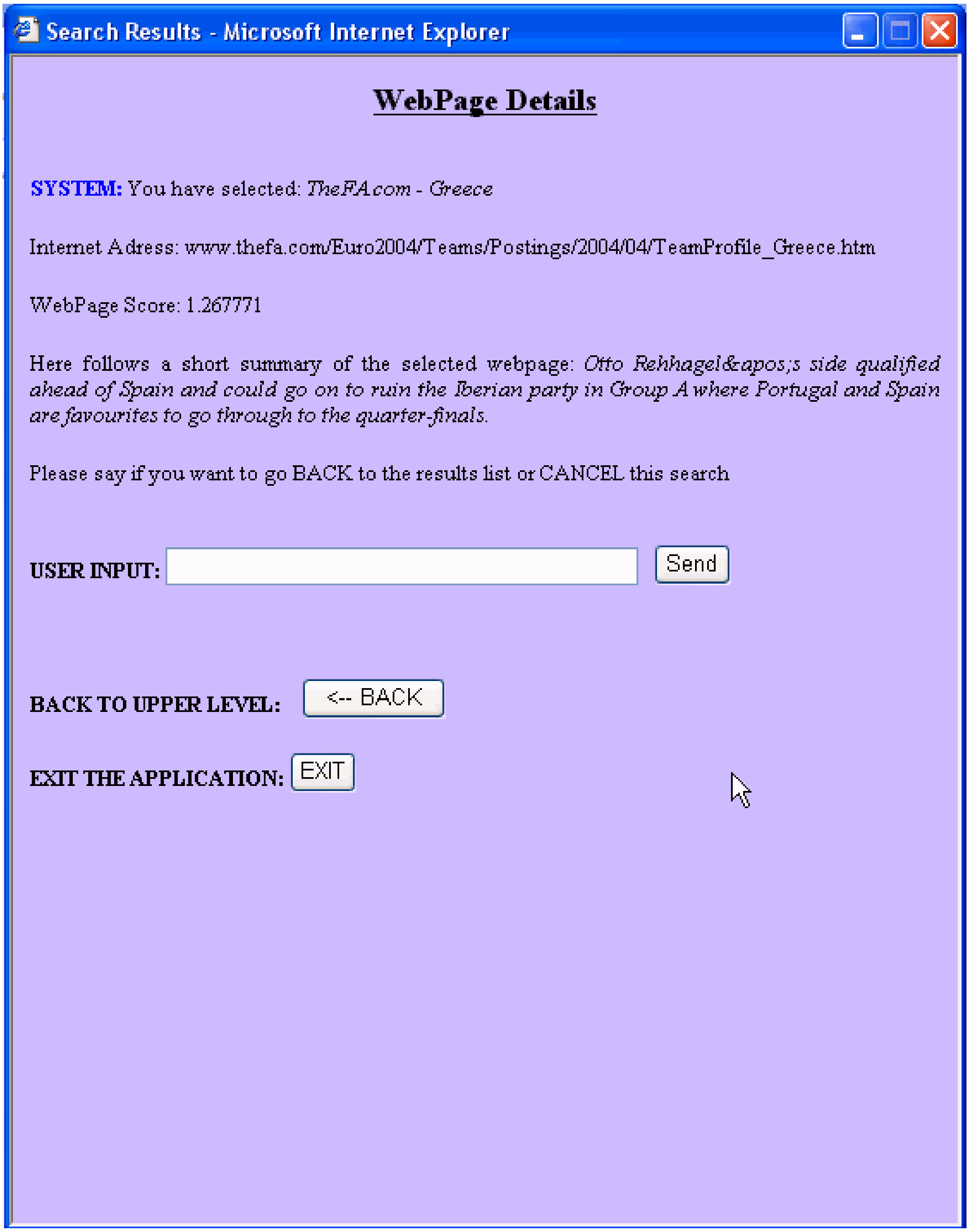
1. TheFA.com - Greece ----> DETAILS
2. TheFA.com - Olympian spirit ----> DETAILS

USER INPUT: select the first result

BACK TO PARENT CLUSTER:

EXIT THE APPLICATION:

11



WebPage Details

SYSTEM: You have selected: *TheFA.com - Greece*

Internet Adress: www.thefa.com/Euro2004/Teams/Postings/2004/04/TeamProfile_Greece.htm

WebPage Score: 1.267771

Here follows a short summary of the selected webpage: *Otto Rehhagel&apos;s side qualified ahead of Spain and could go on to ruin the Iberian party in Group A where Portugal and Spain are favourites to go through to the quarter-finals.*

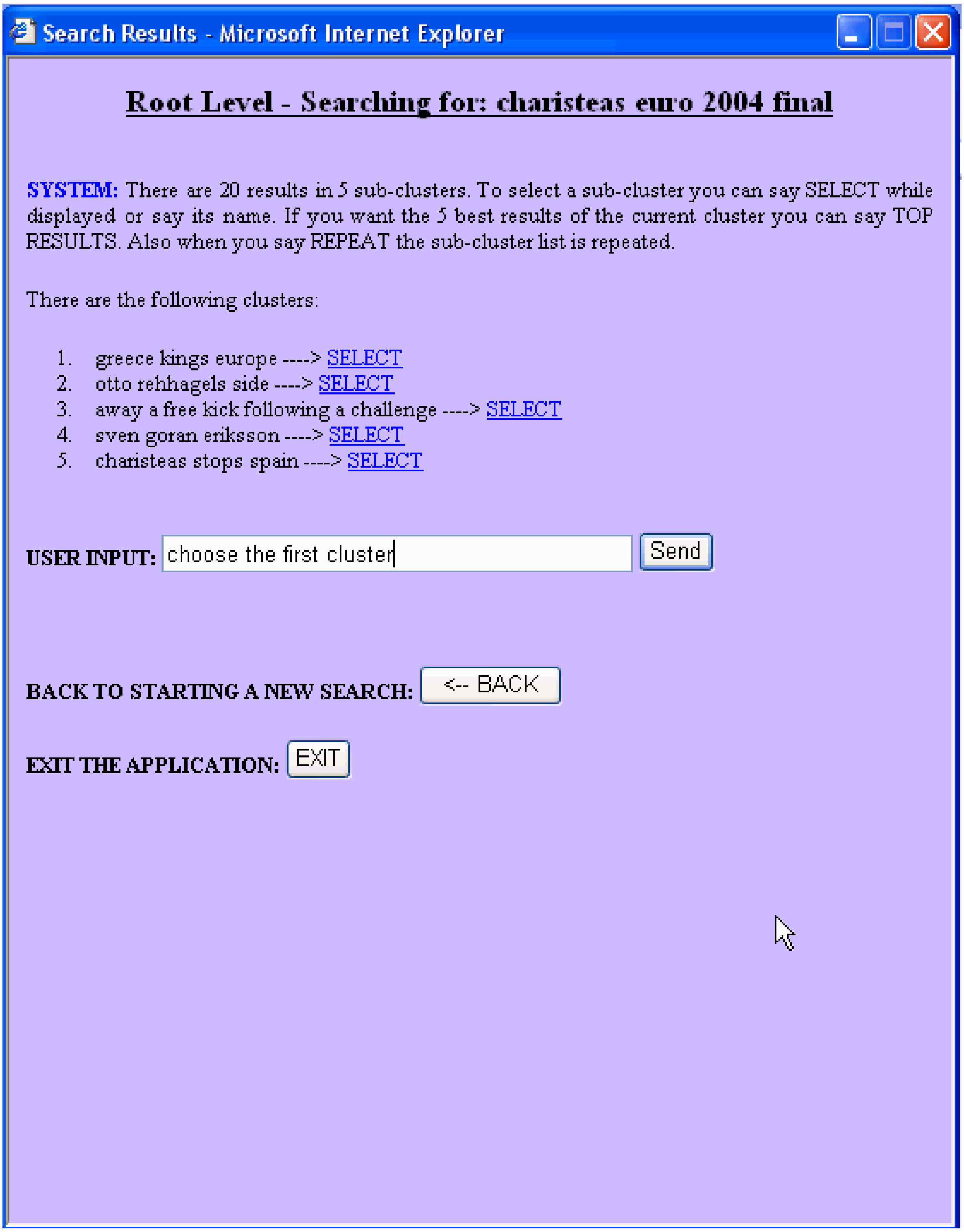
Please say if you want to go BACK to the results list or CANCEL this search

USER INPUT:

BACK TO UPPER LEVEL:

EXIT THE APPLICATION:

12



Root Level - Searching for: charisteas euro 2004 final

SYSTEM: There are 20 results in 5 sub-clusters. To select a sub-cluster you can say SELECT while displayed or say its name. If you want the 5 best results of the current cluster you can say TOP RESULTS. Also when you say REPEAT the sub-cluster list is repeated.

There are the following clusters:

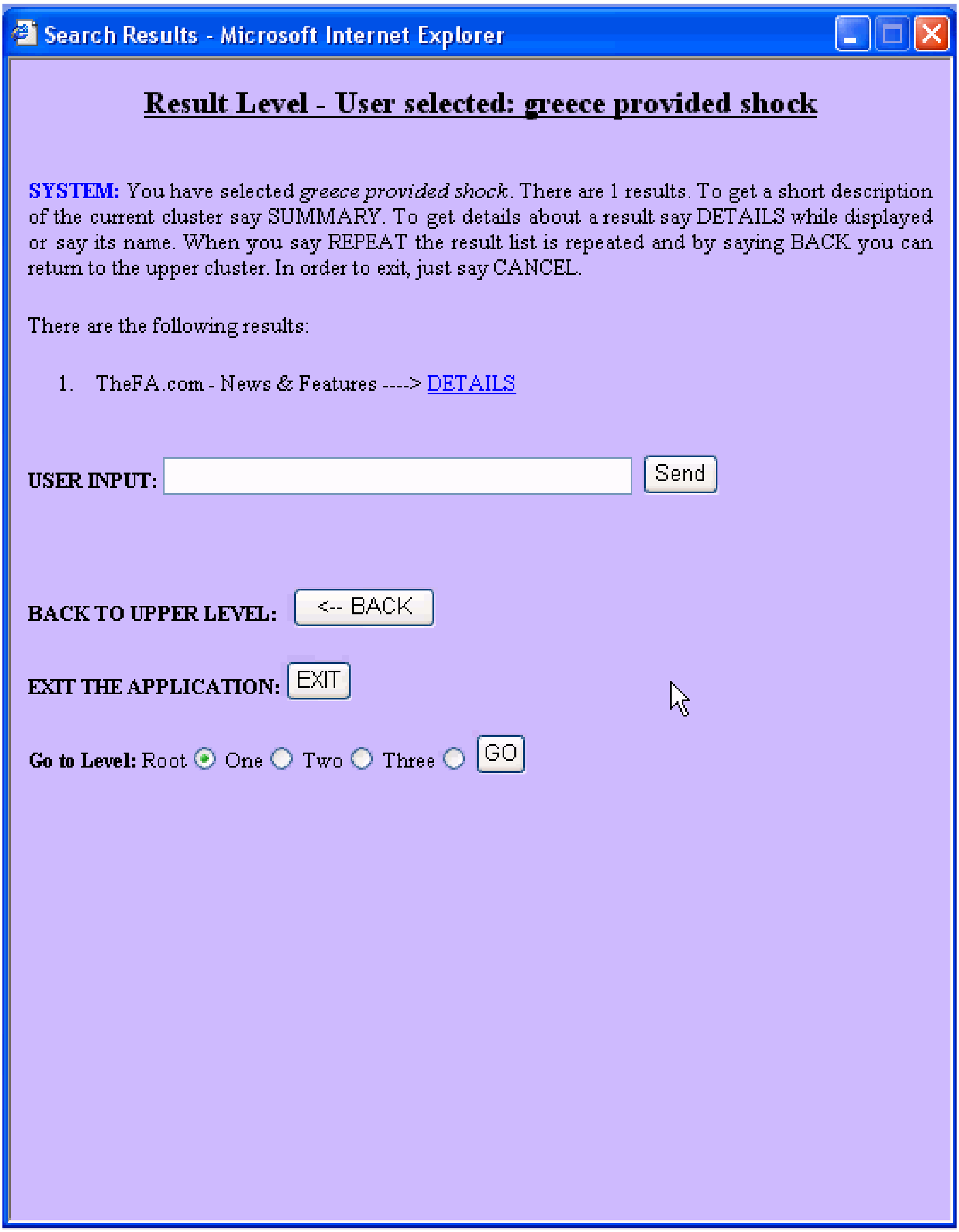
1. greece kings europe ----> SELECT
2. otto rehhagels side ----> SELECT
3. away a free kick following a challenge ----> SELECT
4. sven goran eriksson ----> SELECT
5. charisteas stops spain ----> SELECT

USER INPUT: choose the first cluster

BACK TO STARTING A NEW SEARCH:

EXIT THE APPLICATION:

13



Result Level - User selected: greece provided shock

SYSTEM: You have selected *greece provided shock*. There are 1 results. To get a short description of the current cluster say SUMMARY. To get details about a result say DETAILS while displayed or say its name. When you say REPEAT the result list is repeated and by saying BACK you can return to the upper cluster. In order to exit, just say CANCEL.

There are the following results:

1. TheFA.com - News & Features ----> DETAILS

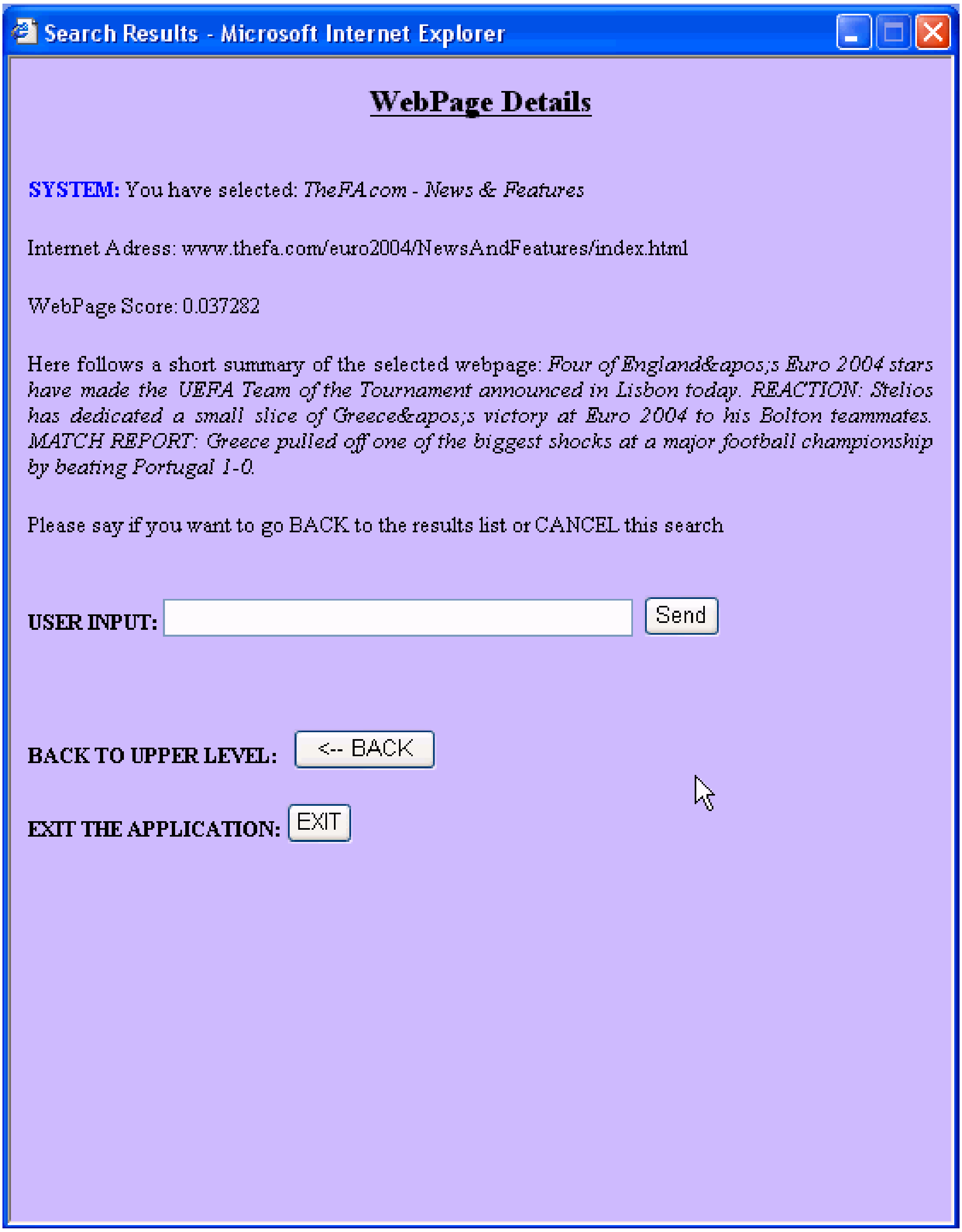
USER INPUT: Send

BACK TO UPPER LEVEL: <-- BACK

EXIT THE APPLICATION: EXIT

Go to Level: Root ● One ○ Two ○ Three ○ GO

14



WebPage Details

SYSTEM: You have selected: *TheFA.com - News & Features*

Internet Adress: www.thefa.com/euro2004/NewsAndFeatures/index.html

WebPage Score: 0.037282

Here follows a short summary of the selected webpage: *Four of England&apos;s Euro 2004 stars have made the UEFA Team of the Tournament announced in Lisbon today. REACTION: Stelios has dedicated a small slice of Greece&apos;s victory at Euro 2004 to his Bolton teammates. MATCH REPORT: Greece pulled off one of the biggest shocks at a major football championship by beating Portugal 1-0.*

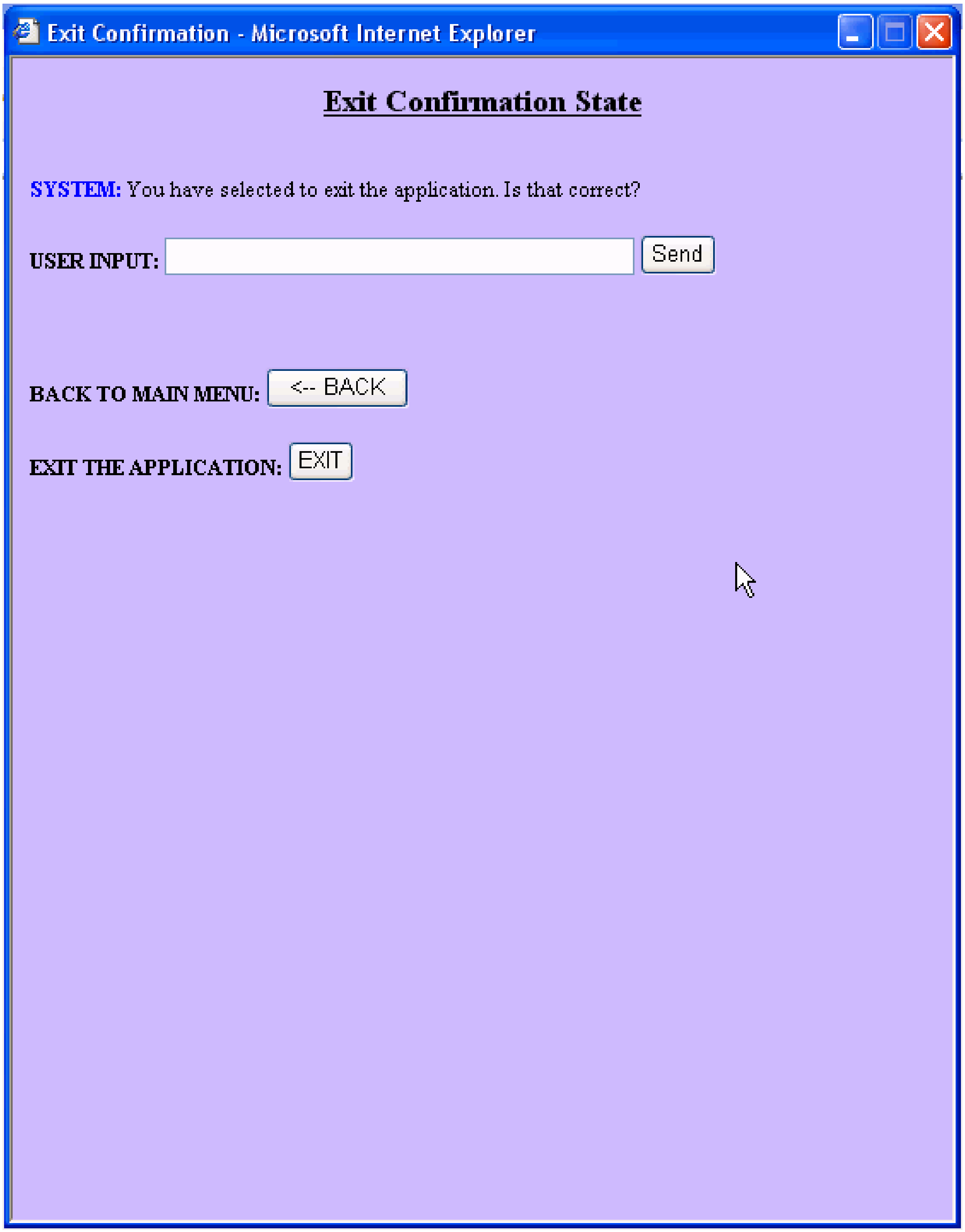
Please say if you want to go BACK to the results list or CANCEL this search

USER INPUT:

BACK TO UPPER LEVEL:

EXIT THE APPLICATION:

15



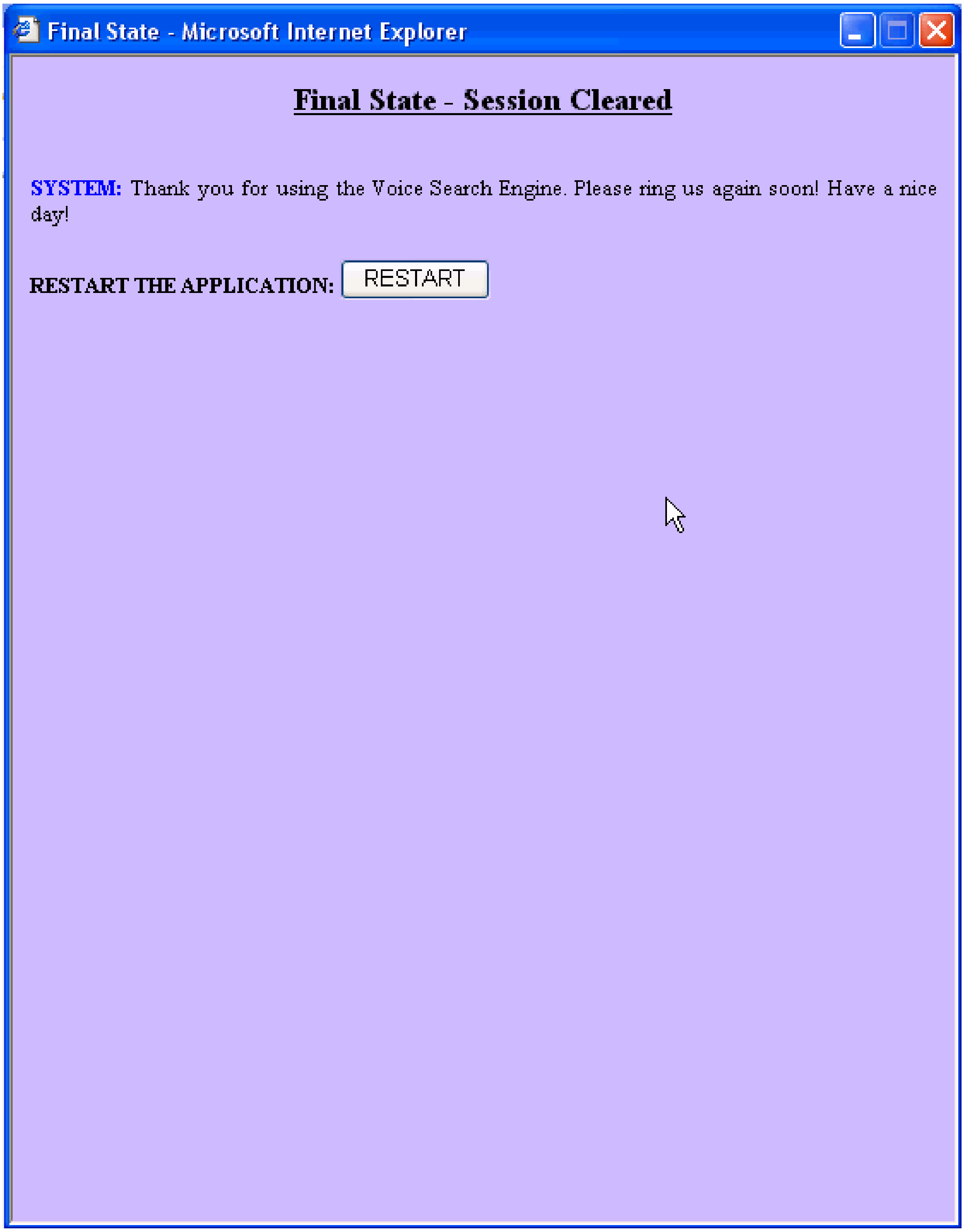
Exit Confirmation State

SYSTEM: You have selected to exit the application. Is that correct?

USER INPUT:

BACK TO MAIN MENU:

EXIT THE APPLICATION:



Final State - Session Cleared

SYSTEM: Thank you for using the Voice Search Engine. Please ring us again soon! Have a nice day!

RESTART THE APPLICATION:

Appendix B

Here we present all the necessary steps one has to take in order to set up the application. We explain how to use the different parts of the source code and how to tune different external applications used.

1) The first thing one should do is to create the repository. To retrieve the web pages any crawler would do. The retrieved pages should be stored under the “repository” directory of our application structure.

2) Having Perl installed is necessary for the back-end to operate. Application tested under Perl 5.8.0

3) After the repository has been created, the web page attributes have to be exported to an index. This is done by the index.sh shell script under the “/back-end/app/bin/interface” directory of the application.

4) Since the back-end is based in relative links, there is nothing else to be done in order to work. However, in order the front-end to be able to see the back-end, the static links of the files in the “/back-end/app/bin/interface” directory should be corrected.

5) The front-end requires Java (tested with java 1.3.1) to be installed. The links in the JSP and servlet source files (regarding the back-end interfaces) have to be corrected. Servlets have to be compiled prior to being used.

6) The dom4j classes have to be included in the CLASSPATH environmental variable.

6) A web server (e.g. Jacarta Tomcat) has to be installed and the JSP and servlet files be placed somewhere the web server can access.

7) The application is accessible, as long the web server is running by calling the corresponding web address through a browser.