

Philippe Mayer

# Improved Search Function for eSarine

Seminar Thesis  
June 2006

Supervised by:  
Prof. Dr. Andreas Meier  
Nicolas Werro



## Abstract

This paper gives an overview of the search function provided by the *eSarine* webshop and then suggests several improvements. Focus is placed on the ranking of results, tolerance for spelling mistakes in queries and the user interface. To this extent, a variety of webshops and theoretical models are compared. The best elements of each are then combined to form a proposal for an improved search function. This recommendation includes a new interface, a ranking function based on the vector space model and an integrated spelling checker that used n-grams count and the Levenshtein-Damerau distance.

**Keywords:** eSarine, Information Retrieval, Approximate String Matching



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Goals . . . . .	1
1.3	Document Structure . . . . .	2
<b>2</b>	<b>Theoretical background</b>	<b>3</b>
2.1	Information Retrieval Systems . . . . .	3
2.1.1	Information Retrieval Systems . . . . .	3
2.1.2	Performance Metrics . . . . .	5
2.2	The Vector Space Model . . . . .	7
2.2.1	Keyword Indexing . . . . .	7
2.2.2	Term Weighing . . . . .	7
2.2.3	Document and Query Representation . . . . .	10
2.2.4	Relevance Calculation . . . . .	11
2.2.5	Limitations of the Vector Space Model . . . . .	11
2.3	Approximate String Matching . . . . .	12
2.3.1	Types of Spelling Errors . . . . .	12
2.3.2	Edit Distance Metrics . . . . .	12
2.3.3	N-gram Analysis . . . . .	14
2.3.4	Soundex . . . . .	15
2.3.5	The Metaphone and Double Metaphone Algorithms . . . . .	16
2.4	The User Interface . . . . .	16
2.4.1	The User . . . . .	17
2.4.2	Input Interface . . . . .	17
2.4.3	Output Interface . . . . .	19
<b>3</b>	<b>Applications to eSarine</b>	<b>21</b>
3.1	eSarine overview . . . . .	21
3.1.1	Internet use by SMEs . . . . .	21
3.1.2	Model-View-Controller architecture . . . . .	22
3.1.3	Internationalisation . . . . .	23
3.1.4	Cross Platform compatibility . . . . .	23
3.2	Ranking Algorithm . . . . .	24
3.2.1	The Current Matching Function . . . . .	24
3.2.2	Improved Matching Function . . . . .	24
3.2.3	Weighing Key Terms . . . . .	25
3.2.4	Ranking Structured Documents . . . . .	26
3.3	The Spelling Checker . . . . .	28

3.3.1	Approximate String Matching Algorithms . . . . .	28
3.3.2	Two Search Strategy . . . . .	30
3.4	Search Interface . . . . .	30
3.4.1	Webshop Users . . . . .	30
3.4.2	The Simple Search Box . . . . .	31
3.4.3	The Advanced Search Page . . . . .	32
<b>4</b>	<b>Conclusion and Outlook</b>	<b>35</b>
4.1	Summary of Findings . . . . .	35
4.1.1	Results Ranking . . . . .	35
4.1.2	Correcting Spelling Mistakes . . . . .	35
4.1.3	The Search Interface . . . . .	36
4.2	Outlook . . . . .	36
<b>A</b>	<b>The Metaphone algorithm</b>	<b>37</b>

# List of Figures

2.1	Search function activity diagram . . . . .	4
2.2	Search results . . . . .	6
2.3	Recurrence relationship for Damerau-Levenshtein . . . . .	13
2.4	Cost calculation for Damerau-Levenshtein . . . . .	14
2.5	Calculating the Damerau-Levenshtein distance . . . . .	14
2.6	A simple search box as seen on oreilly.com . . . . .	18
2.7	Searching amazon.com . . . . .	19
2.8	Advanced search as seen on Ellipse.ch . . . . .	19
2.9	Google spelling suggestions . . . . .	20
3.1	Overview of the MVC pattern in the Struts framework . . . . .	22
3.2	Summary of local weights . . . . .	25
3.3	Summary of global weights . . . . .	25
3.4	eSarine's simple search box . . . . .	31
3.5	The current eSarine advanced search page . . . . .	32



# Chapter 1

## Introduction

### 1.1 Motivation

The Information Systems Research Group of the University of Fribourg (Switzerland) has developed an open source webshop called *eSarine*. It currently features the main features needed for a webshop but still has to be improved in terms of design and functionality.

This project focuses on the search function supplied by *eSarine*. Like the rest of the webshop it offers basic functionality essential to any eCommerce site while leaving room for improvement. The development of a powerful search tool is particularly important as many web surfer judge a website by its search function [17]. This is especially important for webshops as potential customers jump from site to site looking for interesting products [29]. If they fail to find what they are looking for, a competitor's webshop is just one click away.

Currently a basic search function as well as an advanced search page is provided. The basic search function allows the customer to enter a string of characters and the result page then shows all products whose name contains that string. The advanced search page offers the same functionality as well as allowing the customer to define the product type and/or category.

### 1.2 Goals

The aim of this project is to see how the *eSarine* search function can be improved in the following areas:

- Ranking search results in a meaningful manner. As the amount of products available on a webshop increases, the number of results returned is likely to increase proportionally. If the search functions displays several

hundred products that could be of interest to a customer, this information is useless if the user then has to then look through the whole list. The most “relevant” products should be shown first.

- Handling spelling mistakes in query input. Everybody makes spelling mistakes, so the search function should be able to deal with the most common misspellings. For example a customer searching for onomatopeia should not be shown a blank page saying no products found but rather a page containing books on onomatopoeia.
- Providing a user friendly interface. No matter how powerful the new search function is, it won't be of any benefit to potential users if they have to read an instruction manual before they can search for products. It is therefore essential to provide a user friendly and intuitive interface.

### 1.3 Document Structure

Chapter 2 provides a theoretical analysis of the problems this project addresses. It is composed of discussions of various papers that provide theoretical solutions as well as case studies of various other webshops. First a brief overview of the various questions that this project seeks to address is provided in Section 2.1. Performance metrics for algorithms and techniques discussed later in this report are also presented in this section. The vector space model is then introduced in Section 2.2. It is a matching function that would greatly improve *eSarine* search, notably through results ranking. In Section 2.3 various approximate string matching algorithms are discussed in the context of a spelling checker for *eSarine*. Finally, in Section 2.4 use interface guidelines, and various popular webshops are discussed.

In Chapter 3 practical applications of the principles introduced in the previous chapter are presented. Section 3.1 briefly discussed the *eSarine* webshop. Special importance is given to the advantages that *eSarine* attempts to provide over other webshops. The rest of this chapter shows how *eSarine's* search function can be improved in order to better achieve these goals. Section 3.2 shows how the vector space model can be modified and used in order to meet *eSarine's* requirements. Next, in Section 3.3, the approximate string matching algorithms discussed in the previous chapter are evaluated in the context of providing a spelling checker for *eSarine*. Finally, in Section 3.4, improvements to the *eSarine* user interface are discussed.

## Chapter 2

# Theoretical background

### 2.1 Information Retrieval Systems

At first glance, this project addresses three seemingly unrelated problems. The first is how to retrieve information about a product, the second is correcting spelling mistakes and the third is how to rank results. There is however a strong relationship between the three. Information retrieval and string matching both involve searching for information and then ranking the results. In information retrieval, ranking is the process of identifying which set of documents are most likely to be similar in content to a given query. String matching can be described in similar terms, it is the process of identifying which set of strings are most likely to be similar to a given query string [7].

In the case of electronic commerce, the relationship between information retrieval and string matching goes even further. Identifying the correct spelling of a user's input, means finding what possible alternative spelling leads to the product most likely to be of interest to the customer.

#### 2.1.1 Information Retrieval Systems

Before discussing the details of how information retrieval systems work, a brief overview of the general system will be given. An activity diagram of user using such a system is given in Figure 2.1. A brief description of the various steps is given below:

- *Submit Query*: The process of retrieving information is initialised by the user submitting a query.
- *Tokenising and Filtering*: At this stage the query is separated into individual words or tokens. These are then converted to a standard format, e.g. lowercase letters, to allow easy comparisons. So called stop words, that

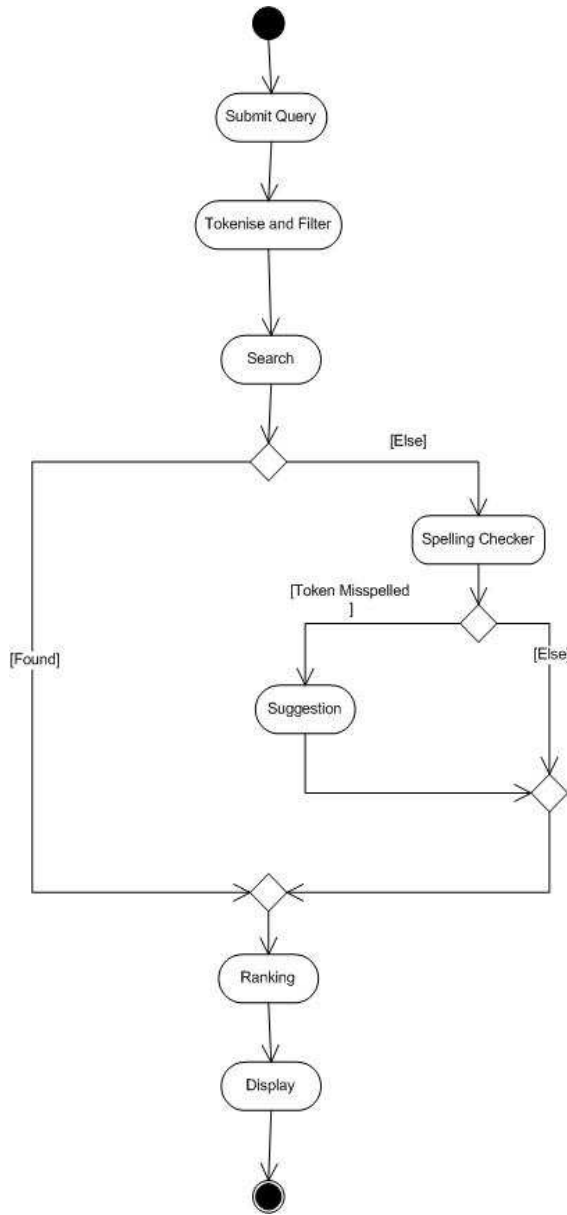


Figure 2.1: Search function activity diagram

occur very frequently such as the words 'and' or 'the' are then filtered out.

- *Search*: Next the database is searched for the individual tokens. This leads to two possible outcomes:
  1. A token is found in the database. In this case the document the term occurs in is passed on to the ranking module.
  2. The token isn't found in the database. This can have two possible causes, either the token is misspelled or it doesn't exist in the database. The token is therefore submitted to the spelling checker.
- *Spelling Checker*: The spelling checker, phonetically or orthographically similar terms are generated. The product database is then searched for these with two possible outcomes:
  1. The token is found. In this case it is noted as being a correction.
  2. The token isn't found. In this case it is discarded

The document is then submitted to the ranking module.

- *Ranking*: Next the retrieved documents are assigned a rank. This is usually a function of the number of query tokens they contain.
- *Display*: Finally search results are displayed in order of their rank. If any of the query tokens were misspelled, appropriate suggestions are made at this point.

## 2.1.2 Performance Metrics

### Relevance

We defined information retrieval as being the process of retrieving relevant data. Relevance is however a subjective notion [30]. This is because a query is only a synthesis of a user's ideas. This means that different users searching for the same key terms have very different concepts of what they are looking for. If we take the example of two users both searching for 'information retrieval', one can be a student doing a project on a subject, and thus be primarily interested in academic publications on the subject. The other can be a professional programmer familiar with the theory but looking for implementation details. In both of these cases, the query was identical, yet the set of documents that each of the two users would consider to be relevant are radically different.

In order to be able to compare different algorithms, it is necessary to define an objective notion of relevance. While this is possible in a controlled environment, it is debatable whether the results obtained are useful to real users.

	Relevant	Non Relevant	
Retrieved	$A \cap B$	$\overline{A} \cap B$	$B$
Not Retrieved	$A \cap \overline{B}$	$\overline{A} \cap \overline{B}$	$\overline{B}$
	$A$	$\overline{A}$	

Figure 2.2: Search results. Taken from [30]

### Recall and Precision

Two common performance metrics for information retrieval system are recall and precision values [20]. The similarities between information retrieval and approximate string matching imply that the same metrics can also be used to evaluate string matching algorithms [7]. Recall is a measure of how many of the total elements in a database are shown whereas precision is a measure of how precise these results are.

Query results can be divided into two overlapping sets, as shown in Figure 2.2,  $A$  represents the set of relevant documents and  $B$  the set of retrieved documents.  $\overline{A}$  and  $\overline{B}$  are their respective complements, namely irrelevant documents and unretrieved results.  $A \cap B$  and  $\overline{A} \cap \overline{B}$  together is the best case scenario, in which all relevant documents, and only relevant documents are retrieved. Realistically however search results are likely to be a mixture of relevant and irrelevant results, which implies trying to minimise the sets  $A \cap \overline{B}$  (relevant documents that are not retrieved) and  $\overline{A} \cap B$  (non relevant documents that are retrieved).

Relevance and precision can then be defined as follows:

$$Precision = \frac{|A \cap B|}{|B|} \tag{2.1}$$

$$Recall = \frac{|A \cap B|}{|A|} \tag{2.2}$$

Traditionally an algorithm  $X$  is said to be superior to an other algorithm  $Y$ , if for every fixed recall value,  $X$  has a higher precision value than  $Y$ . If this is not the case, the average precision values over a given recall span are compared [15].

The ideal combination would be both of high recall and high precision to create an algorithm that returns many precise results. This is however unfortunately impossible. Recall and precision are inversely related [20]. By increasing the number of keywords being searched for and the combining them with the logical AND operator, less results will be found that are more precise. Inversely reducing the number of key terms will increase the number of results returned, at the expense of reducing the quality of these results.

Some kind of balance between quality and quantity must be found. Luckily in the case of information retrieval it is frequently possible to guess what a user

wants. For each search we have to ask ourselves, does the customer want a few precise results, or many vague ones. If a user enters an ISBN number, chances are that they are looking for one and only one book. On the other hand if they are searching for Dolphins, they are likely to want to view as many products as possible on this particular subject. This means that more than one algorithm will be required, depending on the situation.

## 2.2 The Vector Space Model

The vector space model is an attempt to represent natural language documents in a formal manner. This is done by representing individual documents in terms of key words. These terms are then used to form the basis of a vector space, in which documents can then be represented as vectors. Different documents can then be compared to each other for clustering or to queries for searching.

### 2.2.1 Keyword Indexing

Most information retrieval systems in use today rely on keywords to identify the content of documents [27]. It is therefore important to identify certain key words that summarise a document's content. Selecting too many or the wrong key words will dramatically reduce a search functions precision. Selecting too few keywords will reduce recall.

Generally speaking there are three ways of identifying keywords.

1. Manually. This is done by a human operator and thus has the advantage of using human rather than artificial intelligence to classify documents. It is also however very time consuming, and thus expensive.
2. Using Probabilistic techniques. By analysing similarities between documents belonging to the same set, a computer can 'learn' how documents can be classified. Bayesian spam filtering [11] has for instance been applied successfully to filtering e-mails.
3. Rule based automatic filtering. The frequency of a word's occurrences in an article, as well as its position in it can indicate its relative importance. Key words can thus be identified by applying rules to each word in a text.

It is this last technique that will be discussed in detail in the rest of this report.

### 2.2.2 Term Weighing

In order to determine the value of each word in identifying a document, it is a term weighing function is used. Term weights are calculated by considering local and global factors. The general expression for a term weighing scheme is:

$$L_{ij}G_i \quad (2.3)$$

$L_{ij}$  is the local weight for a term  $i$  in document  $j$  and  $G_i$  is the global weight for term  $i$ .

Once the weight for each word in a database has been calculated, the lowest ranked terms, so called stop words which add a lot of volume to a text without adding meaning can be eliminated. Words with a very high term weight can also be eliminated as they appear very rarely and are thus unlikely to appear in any query [30]. The selection of these two cut off points is entirely arbitrary and must thus be found by trial and error.

### Local Weights

The local weight is a function of a terms importance within a given document. Various techniques have been proposed to calculate such a term.

**Binary term weights** The binary term weight is perhaps the simplest local term weight. A term is given the value 1 if it is present within a document and 0 if it is not. Formally expressed this gives:

$$L_i = \begin{cases} 1 & \text{if } f_{ij} > 0 \\ 0 & \text{if } f_{ij} = 0 \end{cases} \quad (2.4)$$

where

$L_i$  is the local weight assigned to a term  $i$

$f_{ij}$  is the frequency of occurrence of the term  $i$  in a given document  $j$

This is equivalent to the standard Boolean model. A given document or query can either be or not be associated with a key term.

**Within document frequency** Common sense tells us that the more frequently a key word appears in a document, the more it tells us about the contents of a document. Each key word is thus assigned a weight proportional to its frequency of occurrence in a document [30]. This is done as follows:

$$L_i = f_i \quad (2.5)$$

where

$L_i$  is the local weight assigned to a term  $i$

$f_{ij}$  is the frequency of occurrence of the term  $i$  in a given document  $j$

This system of weighing is an obvious improvement over binary weighed keywords. Documents can now be ranked according to how closely they can be associated with a given keyword. The downside is that more calculations are required than previously.

**Square root** The square is a relatively new technique for measuring local term weights. Unlike the within document frequency, the square root technique imitates logarithmic methods and attempts to “normalise” a term’s weight. This because a term that appears ten times more frequently than another in a single document is not necessarily ten times more important. Terms are calculated as follows [3]:

$$L_i = \begin{cases} \sqrt{f_{ij} - 0.5} + 1 & \text{if } f_{ij} > 0 \\ 0 & \text{if } f_{ij} = 0 \end{cases} \quad (2.6)$$

where

$L_i$  is the local weight assigned to a term  $i$

$f_{ij}$  is the frequency of occurrence of the term  $i$  in a given document  $j$

### Global Term Weights

In addition to calculating a term’s importance within a give document, it is useful to evaluate how good a term is at differentiating documents. In order to do this, a global weight term is used.

**No global weight term** This is the simplest of all global weighing techniques. Little can be said about it other than that other documents in a collection are not considered and thus no global weight is assigned.

**Inverse document frequency** This is the classic method proposed by Salton. Each occurrence of a word in a document can be assigned a weight equal to its frequency within the document divided by the number of documents that contain that word [10]. Formally this gives

$$G_i = \log \frac{N}{n_i} \quad (2.7)$$

where

$G_i$  is the global weight assigned to a term  $i$

$N$  is the total number of documents in a database

$n_i$  is the number of documents containing the term  $i$  in the database.

This means that a word occurring frequently in one document but rarely in the whole database will be assigned a large weight. Words that are useful for

isolating separate documents in a collection are thus given greater importance. Of all methods seen so far this is the one that requires the most calculations.

**Square root global frequency IDF** The square root global frequency IDF is a relatively new global weight calculating technique that attempts to improve on the already established ones [3]. If the total number of documents in a collection is very small, the inverse document frequency approaches 0. In order to avoid this, the square root global frequency IDF uses the frequency of a words occurrence in the whole database rather than the total number of documents as a numerator. This gives the formula:

$$G_i = \sqrt{\frac{F_i}{n_i} - 0.9} \quad (2.8)$$

$G_i$  is the global weight assigned to a term  $i$

$F_i$  is the total number of occurrences of a term  $i$  in the database

$n_i$  is the number of documents containing the term  $i$  in the database.

### 2.2.3 Document and Query Representation

Once keywords are identified and weighed, documents and queries can be expressed in terms of these. As the name vector space model suggests, this is done in terms of vectors [9]. An  $n$  dimensional vector base is constructed with  $n$  being the total number of keywords indexed.

In order to explain this, it is useful to look at an example. We will consider this report to be a document collection, and the individual chapters to be documents. For the sake of simplicity we will assign only a within document frequency local weight and no global weight. Again for simplicity we will restrict ourselves to the key terms 'eSarine', 'information', 'soundex' and 'query'. In this example, our document collection can be represented as follows:

	Ch1	Ch2	Ch3	Ch4
eSarine	15	1	37	4
information	2	24	11	0
soundex	0	10	4	0
query	1	22	19	3

The keywords 'eSarine', 'information', 'soundex', and 'query' occur with respective frequencies of 15, 2 and 0 and 1 in Chapter 1. This means that Chapter 1 is represented by the vector (15,2,0,1). An obvious and important consequence of this is that a set of documents can be represented as a matrix.

A query can also be represented using a vector. If we want to search for documents containing the words 'eSarine' and 'soundex', our query would be represented as (1,0,1,0).

## 2.2.4 Relevance Calculation

The next stage is to compare our query to each document in the database and then ranking them in order of their similarity to the query. To do this we will once again use the tools of geometry. The traditional choice is to evaluate the cosine of the angle between two different vectors.

The cosine of an angle between two vectors takes on a value between 0 and 1. A value of 1 means that the two vectors are parallel whereas a value of 0 means that they are perpendicular. In terms of information retrieval, this represents the two extremes of identical, or nearly identical, and two entirely dissimilar documents.

Mathematically the cosine of an angle between two vectors A and B can be calculated as follows:

$$\cos \theta = \frac{A \cdot B}{|A||B|} \quad (2.9)$$

In order to better understand how the cosine of an angle between two vectors can be used to compare a query to various documents it is helpful to look at an example. If we continue our previous example and perform the unweighed query (1,0,1,0) on our database, we get the following similarity values using Equation 2.9:

$$\text{Ch1} = \frac{15}{\sqrt{2} \sqrt{230}} = 0.70$$

$$\text{Ch2} = \frac{11}{\sqrt{2} \sqrt{1161}} = 0.23$$

$$\text{Ch3} = \frac{41}{\sqrt{2} \sqrt{1867}} = 0.67$$

$$\text{Ch4} = \frac{4}{\sqrt{2} \sqrt{25}} = 0.57$$

This means that if we rank the results by cosine product we get Ch1 > Ch3 > Ch4 > Ch2.

## 2.2.5 Limitations of the Vector Space Model

A major problem with the vector space model is that it assumes key terms to be independent of each other. As mentioned while describing the document and query model the vectors representing key terms form the basis of our vector space. In order to form a basis, vectors need to be independent. In other words no part of a basis can be expressed in terms of a combination of other members. This is however seldom the case. Terms can be related by synonymity or polysemy.

**Synonymity** This means that several words can have the same meaning. An example of this is information *retrieval* systems and information *searching* systems.

**Polysemy** This means that a word can have more than one meaning depending on the context. An example of this is a car *driver* versus a printer *driver*.

A further problem with the vector space model is that as in the Boolean model, matching is binary. This causes problems if a user mistypes a term or if there are several accepted spellings for a single term, for instance 'centre' and 'center'.

## 2.3 Approximate String Matching

Approximate string matching is a technique for finding similar rather than identical strings. In the context of a search engine this will be used to detect and correct spelling error in users' queries. This can be done because misspelled words are likely to be similar to the correct spelling.

### 2.3.1 Types of Spelling Errors

There are two different types of errors, typographic and orthographic errors. Typographic error occurs when user knows the correct spelling of the word he or she intends to type but mistypes it. These errors are commonly referred to as typing mistakes. Orthographic errors occur when a user has forgotten or doesn't know the correct spelling of a word. These mistakes are particularly problematic as they usually occur when a user attempts to guess the correct spelling of a name and these can have several accepted spellings. Maier, Mayer, Meier and Meyer for instance are all accepted spellings of a name that has just one pronunciation.

As two different types of spelling mistakes exist, two different types of algorithms will have to be considered to correct them. As typographical errors are likely to resemble the correct word, plus or minus the mistyped characters, edit distance algorithms appear at first glance to be the best way to handle them. On the other hand, when a customer doesn't know the correct spelling of a word, they are likely to make a guess and type a word that sounds similar to the correct one, suggesting that cognitive errors are best caught by using a phonetic algorithm.

### 2.3.2 Edit Distance Metrics

The edit distance between two strings is defined as the number of edit operations it takes to transform one string into the other [22]. Edit distances were pioneered by Levenshtein. He proposed that the following edit operations be

allowed: substitution of one letter, deletion of one letter and insertion of one letter. The smaller the edit between two strings is, the more similar they are.

The Damerau-Levenshtein distance algorithm is an improvement of the Levenshtein distance algorithm. It is identical to the original in all aspects other than that it adds transposition or interchanging two adjacent letters as an edit operation. 80 percent of all spelling errors occur within a Levenshtein-Damerau edit distance of 1 of the correct spelling [5]. This explains why this algorithm forms the basis of the Interactive Spelling Checker's<sup>1</sup> near miss detection strategy.

As an example, we can count the number of edit operations necessary to transform the word 'cnsatitoin' into its correct spelling 'constitution'. The operations need are as follows:

1. Insertion of 'o' giving 'consatitoin'
2. Omission of 'a' giving 'constitoin'
3. Substitution of 'u' for 'i' giving 'constitoin'
4. Transposition of 'o' and 'i' giving 'constitution'

By definition, the Damerau-Levenshtein distance between 'cnsatitoin' and 'constitution' is therefore equal to 4.

In order to better understand this technique, it is useful to look at and implementation of it. A well known implementation was proposed by Lowerence and Wagner [31] and is based on previous work by Wagner and Fischer. This is done by filling a  $i$  by  $j$  matrix  $D$ , where  $i$  and  $j$  are the length of the strings being compared, incrementally.  $D[i,j]$  then contains the distance.

The Damerau-Levenshtein distance between two strings  $S$  and  $T$  of length  $i$  and  $j$  respectively can be represented using the following recurrence relationship:

$$DL(0,0) = 0$$

$$DL(i,j) = \min \begin{cases} DL(i-1,j)+1, & \text{insertion} \\ DL(i,j-1)+1, & \text{omission} \\ DL(i-1,j-1)+c(s_i,t_j), & \text{substitution} \\ DL(i-2,j-2)+c(s_{i-1},t_j)+c(s_i,t_{j-1})+1 & \text{transposition} \end{cases}$$

Figure 2.3: Recurrence relationship for Damerau-Levenshtein. Taken from [19]

If the two strings both have a length of zero, the first case, then the distance between them is logically also equal to zero. Otherwise, it is expressed as the total number of operations required to transform  $S$  into  $T$  or vice versa. The strings are compared character by character and at each stage the operation having the minimal cost is retained. For the purpose of this project we will assume that all operations have a cost of 1 or more formally:

<sup>1</sup><http://lasr.cs.ucla.edu/geoff/ispell.html> Visited: 03/05/2006

$$c(s_i, t_j) = \begin{cases} 0 & \text{if } s_i = t_j \\ 1 & \text{otherwise} \end{cases}$$

Figure 2.4: Cost calculation for Damerau-Levenshtein. Taken from [19]

If we apply this recurrence relationship to our previous example, which consisted of comparing the words ‘cnsatitoin’ and ‘constitution’, we get the matrix shown in Figure 2.5. First an empty matrix of size  $i$  (the length of ‘cnsatitoin’) by  $j$  (the length of ‘constitution’) is created. The first row is then filled with integers from 1 to  $j$ . The first column is also filled with integers from 1 to  $i$ . The recurrence relationship in Figure 2.3 is then applied to fill each cell of the matrix. To show how this works we can perform the calculations need to compare  $i$  in ‘cnsatitoin’ and  $o$  in ‘constitution’, this is the value shown in bold in Figure 2.3. From Figure 2.3 we get:

$$DL(i,j) = \min \begin{cases} DL(i-1,j)+1, & \text{insertion} & 4 + 1 = 5 \\ DL(i,j-1)+1, & \text{omission} & 4 + 1 = 5 \\ DL(i-1,j-1)+c(s_i,t_j), & \text{substitution} & 4 + 1 = 5 \\ DL(i-2,j-2)+c(s_i,t_j)+c(s_{i-1},t_{j-1})+1 & \text{transposition} & 3 + 0 + 0 + 1 = 4 \end{cases}$$

The minimum is thus 4 which correspond to transposing the letters  $i$  and  $o$ .

The edit distance between the two strings is the value stored in the bottom right cell.

		c	n	s	a	t	i	t	i	t	o	i	n
	0	1	2	3	4	5	6	7	8	9	10	11	12
c	1	0	1	2	3	4	5	6	7	8	9	10	11
o	2	1	1	2	3	4	5	6	7	8	8	9	10
n	3	2	1	2	3	4	5	6	7	8	9	9	9
s	4	3	2	1	2	3	4	5	6	7	8	9	10
t	5	4	3	2	2	2	3	4	5	6	7	8	9
i	6	5	4	3	3	3	2	3	4	5	6	7	8
t	7	6	5	4	4	3	3	2	3	4	5	6	7
u	8	7	6	5	5	4	4	3	3	4	5	6	7
t	9	8	7	6	6	5	5	4	4	3	4	5	6
i	10	9	8	7	7	6	5	5	4	4	4	4	5
o	11	10	9	8	8	7	6	6	5	5	4	<b>4</b>	5
n	12	11	10	9	9	8	7	7	6	6	5	5	4

Figure 2.5: Calculating the Damerau-Levenshtein distance

### 2.3.3 N-gram Analysis

N-grams are overlapping n-letter sub sequences of a word [13]. For instance the bigrams (n=2) of Lisa are Li, is, sa, while the trigrams are Lis, isa.

The n-grams similarity measure between two words is determined by the number of n-grams that they have in common. One way of calculating this way propose by Justin and Zobel is as follows[6]:

$$\text{gram-dist} = |N1| + |N2| - 2|N1 \cap N2| \quad (2.10)$$

$|N1|$  and  $|N2|$  are the number of n-grams of two different strings  $N1$  and  $N2$  respectively.  $|N1 \cap N2|$  represents the number of n-grams that they have in common. The closer two string resemble each other, the smaller the gram distance.

An illustration of this can be given by calculating the bigrams and trigrams distance between receive and recieve. receive has the following n-grams re, ec, ce, ei, iv, ve numbering 6. recieve also has 6 n-grams re, ec, ci, ie, ev, ve. They share a total of 3 bigrams re, ec, ve. This gives a bigrams distance of  $6+6-2*3=6$ . The trigrams distance can be calculate in the same way. Receive has 5 trigrams rec, ece, cei, eiv, ive as does recieve rec, eci, cie, iev, eve. The two words have just one trigram in common rec, giving a trigrams distance of  $5+5-2*1=8$ .

According to Salton, the best performances are obtained using bi- and trigrams [26]. Monograms are too fine a measure as too many words have single letters in common. Quadgrams or more on the other hand are too coarse as they fail to detect common roots in short words.

### 2.3.4 Soundex

The Soundex algorithm was one of the first attempts at a formalised phonetic matching algorithm. It was developed by Robert Russell and Margaret Odel, and patented in 1918. Soundex uses codes based on the sound of each letter of a string into a canonical form of at most four characters, preserving the first letter [7]. The following description of how Soundex works is given by Knuth [16].

A word is converted into its Soundex key as follows:

1. Retain the first letter of every word, and drop all occurrences of the letters a e h i o u w y in other positions.
2. Convert all remaining letters to a number by using the following rules:
  - b p f v  $\rightarrow$  1
  - c g j k q s x z  $\rightarrow$  2
  - d t  $\rightarrow$  3
  - m n  $\rightarrow$  5
  - r  $\rightarrow$  6
3. Eliminate any adjacent repetitions of code

4. Convert the resulting string to a four letter code either by remove excess code beyond the fourth position or by padding with 0.

As an example, we can calculate the Soundex code of the word Soundex itself. After step 1 we get Sndx. During step 2 this becomes S532. Nothing is done in steps 5 and 6 as we already have a four character code with no adjacent repetition of digits.

A major problem with Soundex is that it doesn't take the differences in pronunciation of letters depending on the ones that surround them, such as 'ph' which is pronounced as 'f'.

Another problem is that the Soundex rules as shown previously are specific to US English pronunciations.

### 2.3.5 The Metaphone and Double Metaphone Algorithms

The Metaphone algorithm was developed by Lawrence Philips and first published in 1990 and addresses the shortcomings of Soundex. The Metaphone algorithms take the different pronunciations of letters and groups of letters into account, depending on their location within words. This does however lead to a much more complex algorithm shown in Appendix A.

As an example, we can encode the word 'Metaphone' itself. After step 1, we get 'Metaphone' as the first letter 'M' maps to itself. After step two, elimination of vowels, we are left with 'Mtpnh'. In step 3, 't' maps to 'T' and 'ph' maps to 'F', giving 'MTF'. This code contains no adjacent repetitions of code and is less than five characters long, so nothing happens in steps 4 and 5.

The original algorithm converted a word into a single key. The problem with this approach was that people of different mother languages pronounce the same word differently. One key is not enough to sum up all this information. This is particularly problematic for *eSarine* as it attempts to work in as many languages as possible.

The Double Metaphone, by the same author was first published in 2000. It gives back two keys for words and names that can plausibly be pronounced in more than one way [24]. The first key represents the US English pronunciation the second the foreign or non English version. Currently alternate pronunciations are supported for name of Italian, Spanish, and French and from various Germanic and Slavic languages.

## 2.4 The User Interface

The interface is the way the user interacts with the search engine. Indeed for many users, the interface is the application. They will never know what

happens in the background, and they probably will never care to know. This is why one approach is to first define an interface that best meets a customer's need and then to build the engine around the constraints imposed by it.

### 2.4.1 The User

The first part of "user interface" is the word "user", thus when setting out to design a user interface it is useful to start by analysing a user's behaviour. When searching a web shop they will probably behave just like they like they do when searching the rest of the web. The following can be said about users in general:

1. Nobody uses advanced search. Every search engine has an advanced search page, but nobody (quantitatively, less than 0.5 % of users) ever goes there [1].
2. Most users enter one or two terms without any Boolean operator and press the 'search' button [20].
3. Most people, after they've done a search, won't look at more than the first few results [1] [2] [23]. This trend has remained fairly constant over the past ten years. The implication is that results must be rank in such a way that items most likely to be of interest to a user are displayed first.
4. A good user interface is one that doesn't force a user to think unnecessarily [17]. This implies that a program shouldn't force a user to make more than the minimum number of decisions [28]. This means that whenever possible, user interface feature should be self evident.
5. "A user interface is well-designed when the program behaves exactly how the user thought it would" [28]. Potential users of our webshop are very unlikely to be first time users of eCommerce sites. They are will have used other online shops and their mental image of how a webshop interface should work is based on their experience with these. It is therefore essential to find out how other webshops work.

### 2.4.2 Input Interface

As mentioned users are very unlikely to use the advanced search page. Most web shops such as Amazon<sup>2</sup>, O'Reilly<sup>3</sup>, Ellipse<sup>4</sup> don't even have one. Those inclined to use advanced search features are much more likely to search online library catalogues such as the "Netzwerk von Bibliotheken und Informationsstellen in der Schweiz" <sup>5</sup> or online film databases such as the Internet Movie

---

<sup>2</sup><http://www.amazon.com> Visited: 06/05/2006

<sup>3</sup><http://www.oreilly.com> Visited: 06/05/2006

<sup>4</sup><http://www.ellipse.ch> Visited: 06/05/2006

<sup>5</sup>URL: <http://www.nebis.ch> Visited: 05/06/2006

Database<sup>6</sup>. This indicates that while an advanced search page can be provided for some users, it would be better focus on improving the search box.

### The Simple Search Box

The simple search box is ubiquitous; all online shops mentioned previously have one. It is always displayed prominently at the top of a webpage, and always consists of a text box labeled 'search' with a button labelled 'go' on the right. This can be seen in Figure 2.6. Customer use the search box by typing a query and then either pressing enter or pressing the 'go' button. As such they correspond to what most users want to do.



Figure 2.6: A simple search box as seen on oreilly.com

Search engine users make a number of assumptions when using the simple search box. These include the assumption that all product fields are searched [20]. This is an important feature offered by the search box on all webshops surveyed. Amazon was one of the first webshops whose search function did not require a customer to specify what product field should be searched [17]. Amazon's success shows that users don't want to have to restrict their search to certain fields.

### The Semi Advanced Search Box

A compromise between the simple search box and the advanced search page is a search box that allows the user to specify some search parameters. Amazon for instance allows the customer to easily specify whether all products should be searched or just a particular category. This can be seen in Figure 2.7. This is very intuitive and thus helps improve search results without increasing complexity. This solution also makes sense as it is safe to assume that the customer has at least some idea of the product that they are looking for. Problems can however occur if a product can exist in two or more categories without the customer caring which one its in. For example if someone is searching for a film, it can be under the DVD or VHS section.

### The Advanced Search Page

When discussing user behaviour, I mentioned that very few people use the advanced search page offered by search engine. It is therefore hardly surprising that many webshops don't even have one. One of the rare online shops that

---

<sup>6</sup>URL: <http://www.imdb.com> Visited: 05/06/2006



Figure 2.7: Searching amazon.com

does provide an advanced search page is that of Librairie Ellipse shown in Figure 2.8.

A key feature of this advanced search page is its use of tabs. When a user selects a tab, it jumps to the front. Selecting the DVD tab in Figure 2.8 for instance, brings the DVD page to the front. What differentiates tabs from ordinary buttons is that tabs are a physical metaphor. They look like and act like physical tab dividers use as filing separators. Because of this tabs are very intuitive: when Microsoft first began performing usability tests on tabs, usability was at close to 100% [28]. This means that nearly everyone tested understood how tabs work.

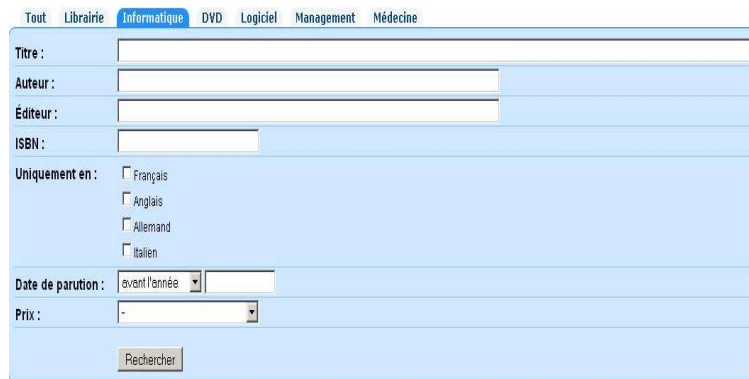


Figure 2.8: Advanced search as seen on Ellipse.ch

### 2.4.3 Output Interface

There are two ways of integrating spelling checkers into the output interface. One way of doing this is by transparently correcting mistakes and searching for

similar terms. This can be problematic if a user is searching for a very rare and specific word. If the word is “corrected” transparently, the customer will never be shown the information they were looking for and more importantly will never be given the chance to force the computer to look for the exact spelling. The alternative is to perform the search according to the original query and then suggesting an alternative spelling. It would also be a good idea to limit the suggest to one or two words as we have already established that search engine user don’t like to look through long lists of results. An example of such an interface is that of Google shown in 2.9.



Figure 2.9: Google spelling suggestions

## Chapter 3

# Applications to eSarine

### 3.1 eSarine overview

The Information Systems group of the University of Fribourg has developed an open source webshop called *eSarine* aimed at medium sized enterprises which want to use the Internet as an additional distribution channel [4]. Key features include [4] [21]:

- Separation of data from business logic and presentation via the Model View Controller design pattern.
- Support for internationalisation built into the Struts framework.
- Low Cost notably via integration with existing solutions thanks to the cross platform compatible Java run time environment.

#### 3.1.1 Internet use by SMEs

A market study [14] done by the University of Fribourg has shown a dramatic increase in Internet use by small and medium sized enterprises (SMEs). In 2000 80% of businesses with more than 20 employees used the Internet. For business with more than 100 employee 90% made use of the Internet. In the cases of businesses with less than 20 employees, the percentage was significantly lower. In 2002, global Internet use by businesses had increased, particularly for businesses with less than 5 employees where Internet use increased from 40% in 2000 to 80% in 2002. This trend has continued in recent years and as of 2005 95 % of SMEs in Europe are connected to the Internet and between 58 and 82% have their own website. Far fewer, SMEs however (only 16%) sell online [8].

Opening an electronic commerce site is particularly challenging for SMEs. Significant costs include: hardware costs for servers, internet connection costs,

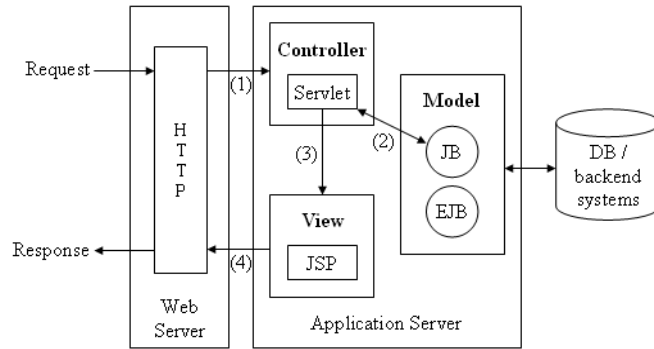


Figure 3.1: Overview of the MVC pattern in the Struts framework

software licensing costs and technical know-how required to run the web-shop. *eSarine* attempts to address each of these points to the extend possible.

### 3.1.2 Model-View-Controller architecture

Most interactive web applications, including *eSarine*, implement the following cycle:

1. display a HTML page containing forms that request input from a user
2. treat and save the data entered by the user
3. display the results on a new page

The Apache Struts framework, on which *eSarine* is based offers a general support for these features. Struts was initially developed to extend the Servlet architecture to encourage a developers to adopt the Model-View-Controller(MVC) design pattern. A overview of this is provided in Figure 3.1.

**Model** The model represents state of an application. It is independent of the view and the controller. In the case of *eSarine* the model is the data describing products. Typically this includes a products name, authors, reference number, and category or sub category that it belongs to. Additionally images and other resources are also stored in the database. This is done in order to facilitate the store administration. If these where stored as separate files, ftp access to the server would be necessary. Instead everything can be configured from the same web based interface.

**View** The view is the way the model is presented to the user. As the view can only read the model, but not modify it, it has to be informed whenever the model changes. In the case of *eSarine* this is done using Java Server Pages (JSPs). A JSP can contain HTML tags as well as call to Java programs. This means that the content of web pages is dynamically generated.

**Controller** The controller determines what actions should be performed in terms of a user's input. In the case of *eSarine* this is done using Servlets.

A key advantage of the MVC design pattern is the clean separation of data and logic that it offers. This differentiates *eSarine* from other open source webshops based on PHP which tend to mix application logic and data. This is particularly important for multilingual support where the model varies depending on the language, but the view and controller must remain identical.

### 3.1.3 Internationalisation

An important part of *eSarine* is its support for multiple languages. This is especially important in a country like Switzerland where webshops have to deal with customers with four different mother tongues. At the database level product fields can be stored in different languages. The language the web pages are displayed in is determined by analysing the locale string sent by a customer's web browser. By default support is provided for English German and French, additional languages can also easily be added by a webshop administrator.

### 3.1.4 Cross Platform compatibility

The high costs and technical know-how represent particular challenges for SMEs wishing to enter online business. *eSarine* addresses both of these concerns. Not only is the interface easy to use, but it also represents a cost effective solution. The webshop itself is licensed under the General Public License<sup>1</sup> and can thus not only be obtained free of charge but can also be easily tailored to a particular business's needs.

Additionally the components needed to run the webshop can be obtained free of charge. Many of these are open source, such as the Apache<sup>2</sup> web server, the Tomcat<sup>3</sup> or Jboss<sup>4</sup> Servlet containers, and the PostgreSQL<sup>5</sup> and MySQL<sup>6</sup> databases.

Since *eSarine* is written in Java, it benefits from the latter's cross platform capabilities. This means that companies can leverage existing infrastructures.

---

<sup>1</sup><http://www.gnu.org/copyleft/gpl.html> (Visited: 19/07/2006)

<sup>2</sup>Available at <http://www.apache.org/> (Visited: 28/4/2006).

<sup>3</sup>Available at <http://tomcat.apache.org/> (Visited: 28/4/2006).

<sup>4</sup>Available at <http://www.jboss.com/> (Visited: 28/4/2006).

<sup>5</sup>Available at <http://www.postgresql.org/> (Visited: 28/4/2006)

<sup>6</sup>Available at <http://www.mysql.com/> (Visited: 28/4/2006)

The advantage of Java's write once read everywhere ability is that eSarine can run on any operating system provided that the appropriate Java virtual machine and an EJB container are available.

The Java Database Connectivity (JDBC) API is a set of classes that provide a mechanism for submitting SQL statements to different databases [12]. In addition to this eSarine provides wrapper classes for the following databases: IBM DB2<sup>7</sup>, Microsoft SQL server<sup>8</sup>, MySQL, Oracle<sup>9</sup> and PostgreSQL. As the eSarine source code is freely available, users who require support for another database can easily add their own wrapper classes.

## 3.2 Ranking Algorithm

### 3.2.1 The Current Matching Function

Like many information retrieval systems today [10], the *eSarine* search engine uses the Boolean model, and this despite its limitations. Its foundations lie on Boolean algebra and it is frequently implemented using set theory. Its main drawing points are its simplicity and its power of expressiveness. Any database query can be expressed in terms of relational algebra operators [18]. Its main draw back is that being founded on the notion of crisp sets. It can only deal with crisp queries on which binary matches can be performed.

As mentioned in the introduction, a user's request for information will only ever be a synthesis of an idea they have. In some case this can be translated directly into a Boolean query. This is true for instance if a user wants to see all products that are compact discs. This set of products is a crisp one, products are either a CD or they are not, and all CDs are equally CD.

In most cases however, customers will be searching for concepts that cannot easily be translated into sharp sets. This is for instance the case if a user is searching for 'dolphins' for instance. Some products are clearly more about dolphins than others. A book dedicated to the subject is more likely to interest the user than one which contains only a chapter on the subject.

### 3.2.2 Improved Matching Function

The vector space model was introduced in Section 2.2 as an "ideal" matching function. As mention in Section 2.2, the search for documents can be decomposed into three action: representing documents in term of key terms, representing queries in term of key terms, ranking documents according to relevancy to the query. These three steps will now be discussed in the context of electronic commerce.

---

<sup>7</sup>Available at <http://www-306.ibm.com/software/data/db2/> (last visited: 28/4/2006)

<sup>8</sup>Available at <http://www.microsoft.com/sql/> (last visited: 28/4/2006)

<sup>9</sup>Available at <http://www.oracle.com/database/> (last visited: 28/4/2006)

Name	Formula
Binary	1 if $f_{ij} > 0$ 0 if $f_{ij} = 0$
Within document frequency	$f_{ij}$
Square root	$\sqrt{f_{ij} - 0.5} + 1$ if $f_{ij} > 0$ 0 if $f_{ij} = 0$

Figure 3.2: Summary of local weights

Name	Formula
No global weight	1
Inverse document frequency	$\log\left(\frac{N}{n_i}\right)$
Square root global frequency IDF	$\sqrt{\frac{E_i}{n_i} - 0.9}$

Figure 3.3: Summary of global weights

### 3.2.3 Weighing Key Terms

Representing documents in term of key terms has two advantages, namely faster search time and documents and queries that are represented in the same form. The advantage of the first point is obvious, rather than searching through a whole document, only key terms need to be searched. The second point can be used to extend the existing search function by allowing a user to indicate interesting documents and then search for similar ones.

As mentioned in Section 2.2.1, key term weights are a combination of documents specific values (how important a term is in describing a document) and global values (how useful a key term is in identifying a particular document). A summary of the local and global terms discussed in Section 2.2.1 is provided in Figures 3.2 and 3.3 respectively.

There are several criteria for selecting local and global terms that are specific to *eSarine* and other web shops,

1. Queries are likely to be very short (one or two words).
2. Product descriptions are relatively short (less than a paragraph) and are either written by, or at least controlled by the webshop owner.
3. *eSarine* is targeted at small and medium sized enterprises.

The first point eliminates any form of term weighing on the query. The within document frequency and square root local weighing schemes both assume that different terms have different frequencies of occurrence. This implies that at

least some query terms must be repeated. With an average query length of 1.3 this does not seem possible.

The second point means that many of the disadvantages of the within document frequency local weighing technique disappear. Because this technique assigns each key terms a weight that is directly proportional to it's frequency of occurrence, it naturally favours long documents over short ones and it is vulnerable to so called key word spamming. Neither of these two cases occur in the context of webshops so we don't need to resort to more complex techniques.

The third point suggests that it would be a good idea to reduce the complexity of search and index methods, in order to increase search speed and thus reduce the need for expensive hardware. All the global weighing methods survey are calculation intensive because all documents in a database need to be taken into account when calculated the weight for a key term. This suggests that if possible, global weights should be ignored when calculating the relevancy of a particular product.

### 3.2.4 Ranking Structured Documents

Information describing products is stored in several fields. Not all of these are equally useful in identifying a product. If a user's query matches a product's ISBN or ASIN for instance, this is probably the one and only relevant product. A product's title and author are also good indicators of a products relevancy. A products title usually contains terms that accurately describe its central theme. A products description on the other hand is much worse at identifying a product. The description can for instance contain sentences as "inspired by \$other\_author" yet that doesn't make this particular product relevant to queries for \$other\_author.

One way to take the relative importance of different fields into account is to calculate the similarity between the query and each field and then to calculate a weighed average. Formally this gives:

$$sim(Q, D) = \sum_{i=1}^n W_i sim(Q, S_i) \quad (3.1)$$

where

$sim(Q, D)$  is the similarity between query  $Q$  and document  $D$

$W_i$  is the weight of section  $S_i$

$sim(Q, S_i)$  is the similarity between the query  $Q$  and the section  $S_i$

$n$  is the number of fields used to describe a document.

In order to illustrate this we will return to the example given in Section 2.2. In that example documents were treated as a homogeneous collection of words all equally important. We will now break the documents down into three fields, chapter title, section headings and content.

If we consider only the chapter titles, we get the following matrix:

	Ch1	Ch2	Ch3	Ch4
eSarine	0	0	1	0
information	0	0	0	0
soundex	0	0	0	0
query	0	0	0	0

If we perform the same query as before, namely 'eSarine soundex' we get the following similarity measure according to Equation 2.9:

$$\text{Ch1} = \frac{0}{\sqrt{2} \sqrt{0}} = 0$$

$$\text{Ch2} = \frac{0}{\sqrt{2} \sqrt{0}} = 0$$

$$\text{Ch3} = \frac{1}{\sqrt{2} \sqrt{1}} = 0.7$$

$$\text{Ch4} = \frac{0}{\sqrt{2} \sqrt{0}} = 0$$

If we now consider the individual section headings we get:

	Ch1	Ch2	Ch3	Ch4
eSarine	0	0	1	0
information	0	1	0	0
soundex	0	0	0	0
query	0	0	0	0

The query for 'eSarine soundex' clearly give the same results as for the chapter titles.

Finally we will perform the same operations for the content of the individual chapters. The resulting matrix is:

	Ch1	Ch2	Ch3	Ch4
eSarine	15	1	35	4
information	2	23	11	0
soundex	0	10	4	0
query	1	22	19	3

Comparisons to the query 'eSarine soundex' according to Equation 2.9 give:

$$\text{Ch1} = \frac{15}{\sqrt{2} \sqrt{230}} = 0.70$$

$$\text{Ch2} = \frac{11}{\sqrt{2} \sqrt{1114}} = 0.23$$

$$\text{Ch3} = \frac{39}{\sqrt{2} \sqrt{1867}} = 0.66$$

$$\text{Ch4} = \frac{4}{\sqrt{2} \sqrt{25}} = 0.57$$

If we consider the chapter title to be very useful at describing it, the section heading less useful and the content even less so, we can assign them relative weights of 0.5, 0.4 and 0.1. According to Equation 3.1 we get:

$$\text{Ch1} = 0.5 * 0 + 0.4 * 0 + 0.1 * 0.7 = 0.07$$

$$\text{Ch2} = 0.5 * 0 + 0.4 * 0 + 0.1 * 0.23 = 0.023$$

$$\text{Ch3} = 0.5 * 0.7 + 0.4 * 0.7 + 0.1 * 0.66 = 0.698$$

$$\text{Ch4} = 0.5 * 0 + 0.4 * 0 + 0.1 * 0.57 = 0.057$$

The final ranking of The chapters of this report is thus Ch3 > Ch2 > Ch1 > Ch4.

If we compare this result with the one obtained in Section 2.2 which was Ch1 > Ch3 > Ch4 > Ch2, we can immediately see that Chapters 2 and 3 are now rated higher whereas Chapters 1 and 4 are rated lower. This can be explained by the fact that Chapters 1 and 4, the introduction and conclusion respectively provide a summary of the whole report and thus contain the term 'eSarine' with a relatively high frequency. Thus if we consider all parts of a document to be of equal value, these Chapter are rated highly. If on the other hand the value of Chapter and section headings is increased, Chapters 2 and 3 which discuss *eSarine* and Soundex in some detail are given greater importance. As these two chapters deal with theoretical solution, including Soundex, and applications to *eSarine*, the new ranking is much more accurate.

### 3.3 The Spelling Checker

Various selection criteria need to be considered when choosing matching algorithms. Speed, precision and recall are all important. Additionally, as *eSarine* is designed to support multiple languages, the spelling checker must also be language independent, or at least easily adaptable to several languages.

#### 3.3.1 Approximate String Matching Algorithms

##### Phonetic Algorithms

Three different Phonetic Algorithms were introduced in Section 2.3. These were the Soundex, Metaphone and Double Metaphone. The main advantage of phonetic algorithms is their speed.

A big advantage of phonetic algorithms is their speed. Spelling Checking techniques that rely on such techniques are fast because the phonetic codes of key terms stored in a database can be pre calculated. This means that when a user performs a search, only the phonetic code of the query terms needs to be calculates.

A major disadvantage of phonetic algorithms is that they provide binary matches, that is words are either phonetically identical or not, this means candidates cannot be ranked. Additionally, if a word is mistyped in a way that changes its pronunciation, its phonetic code will be different. 'decipher' for example has a Metaphone code of TSFR whereas 'decihper' maps to TSPR.

Because identical word combinations are pronounced differently in different languages, phonetic algorithms are limited to a single language. This is particularly problematic for *eSarine* which attempts to provide a multilingual web-shop.

Lait and Randell have compared recall and precision values and have come to some surprising conclusions. They found that simple algorithms such as Soundex are not only faster but also offer better performance than more modern algorithms such as Metaphone [25].

### **Edit Distance**

A single edit distance measure, the Levenshtein-Damerau distance was introduced in Section 2.3. While much slower than phonetic algorithms, it offers higher quality results.

The big advantage of edit distance metrics is the quality of the results obtained. As mentioned, in Section 2.3, 80% of all spelling errors fall into one of the four types of mistakes covered by the Levenshtein-Damerau distance. These four categories of errors were: substitution of one letter, deletion of one letter and insertion of one letter and exchange of two adjacent characters.

This increase in quality does however come at the expense of longer execution time. Unlike phonetic codes which can be calculated in advance and stored, the edit distance must be calculated at each comparison which compromises performance. This naturally greatly increases the amount time taken to find possible correct spellings.

### **N-Grams Count**

The n-grams count provides an intermediate solution both in terms of speed and effectiveness.

Like phonetic codes, the n-grams of documents in a database can be calculated before a search is made. This means that n-grams count is faster than edit distance measures. As a single term has many n-grams, but only a single, or at most two, phonetic code, string comparisons techniques relying n-grams count are slower than those based on phonetic codes.

A comparison of various algorithms has shown that the n-grams count is more effective at find correct spelling of a misspelled word than the Soundex algorithm [19]. The same study however also showed it to be less accurate than the edit-distance.

### 3.3.2 Two Search Strategy

Given that a complete search of a lexicon is prohibitively expensive, Zobel and Dart suggest that some form of indexing be used to first extract likely candidates from the lexicon, in a coarse search. These candidates can then be processed in more carefully in a subsequent fine search [6]. When performing the first search the emphasis must be on maximising recall even at the expense of precision. The reason for this is to avoid eliminating what might finally be the best candidate. A second search is then performed on these candidates this time maximising precision, so as to find the most likely correct spelling.

#### The Coarse Search

A traditional choice for this task are phonetic coding such as the Soundex method [6]. As mentioned previously, in the case of *eSarine* however the speed of phonetic algorithms is negated by the fact that they are tied to a single language. Since edit distance is too slow, the only remaining choice is the n-grams count.

#### The fine search

By process of elimination, the only algorithm that can logically be used for fine searches is the edit-distance comparison. Phonetic algorithms were already discarded for the coarse search as they were too coarse, and we have already used the n-grams count.

## 3.4 Search Interface

Currently the *eSarine* webshops supports two different search interfaces, a simple search box and an advanced search page.

### 3.4.1 Webshop Users

Just as we began our discussion of user interfaces by analysing user behaviour, it is useful to look at webshop users' behaviour before discussing webshop interfaces. Webshop users fall into several categories:

**The online surfer** One group includes those who don't know exactly what they want and thus want to "window shop". They tend to roam the Web aimlessly jumping from one website to next [29]. Upon reaching website some of these visitors will immediately hunt for a box with the word 'search' written next to it [17].



Figure 3.4: eSarine's simple search box

**The online consumer** Some online surfers become consumers. Online consumers tend to interactively browse the product catalogue a webshop offers [17]. Unlike the online surfer, the consumer will have more than a superficial notion of what they are looking for and thus it makes sense to offer them the possibility of restricting their search to certain areas via an advanced search page.

**The online key customer** The online customer is a user that regularly shops at a certain webshop [29]. Unlike first time visitors, they will have a certain amount of knowledge of *eSarine* and its advanced features. This means that they are the group of user most likely to use the advanced search page.

### 3.4.2 The Simple Search Box

As mentioned in Section 3.1 providing a user friendly interface is priority of *eSarine*. In Section 2.4 we established that most users only use the simple search box. This means that most of the effort that is put into improving the search function should go into improving this aspect as it will provide benefits of the largest section of users.

#### The Current eSarine Search Function

The simple search box provided by *eSarine* is shown in Figure 3.4. Customers enter query terms into the search field and then press the search button. The query terms are then compared to all product titles. The products whose title contain the query terms are then displayed. Thus the simple search box currently provided by *eSarine* offers the following "features": only the product title fields are searched and a user can not restrict his or her search to a particular category of products

#### Possible Improvements

**Extended search** None of the other webshops surveyed in Section 2.4 restrict a user's query to a products title. In Section 2.4 we discovered that many popular webshops search all product categories by default, and that a users mental image of how a product is supposed to work is formed by their prior

experience. It would therefore be a good idea to extend the search to include all product fields by default.

**Allow users to reduce search scope** As to the second point, it is reasonable to assume that at least some of the customers using the simple search box know what type of product they are looking for. By reducing the amount of information being searched through, search performance can also be improved. If the concept of a search box such as the one available at Amazon's web-site, shown in Figure 2.7, is implemented, by default the whole product database is searched, while allowing a customer to restrict their search to a certain product category.

By proposing more and more restrictive sub categories, part of the functionality of the advanced search page is offered to the vast majority of users that use only the simple search box. Another advantage is that it eases the transition from online surfer to online consumer. The casual surfer can start off by entering a vague product description before refining their search.

### 3.4.3 The Advanced Search Page

#### The Current eSarine Advanced Search Page

The advanced search page provided by *eSarine* allows customer to specify which product field should correspond to which query terms. As can be seen in Figure 3.5, customers can restrict searches to a particular category or product type. If a category or product type is specified but no product name is specified, a user is given the possibility of searching category or product type specific fields.

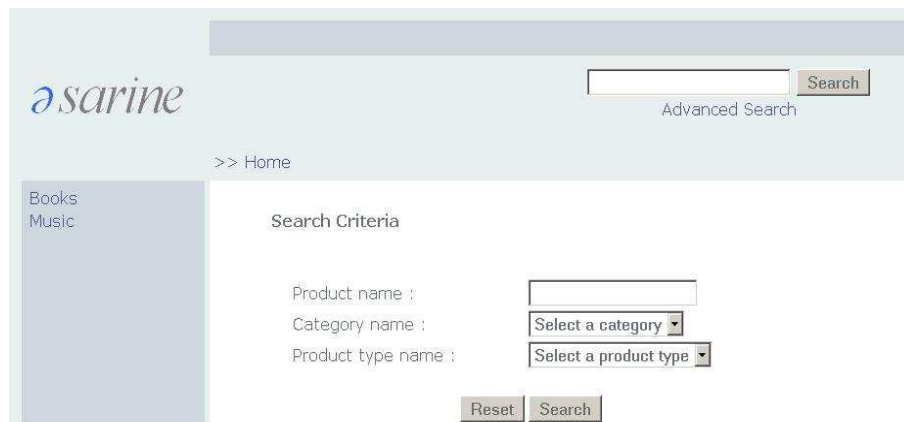


Figure 3.5: The current eSarine advanced search page

From an user interface point of view, the advanced search page suffers from two deficiencies. Firstly the category field lists both categories and sub categories. This can be confusing for users, especially if a webshop offers a large selection of products divided into tens if not hundreds of categories and subcategories.

Secondly product type specific fields are only searchable if a user specifies a product type. The problem with this approach is that a product's category and type overlap. 'book' for instance is both a category and a type. As the category selection is placed above the type selection, a user can select 'book' as a category and then leaving the type selection empty, as this selection is redundant. In this case the user will however not be redirected to the 'book' specific search page.

### **Possible Improvements**

**Tab Browser Interface** One solution would be to implement a tab based interface as shown in Section 2.4. The tabs would represent major product categories and their corresponding page would contain category specific fields. Because major product categories usually correspond to product types, all reference to the latter can then be removed. This would eliminate any ambiguity.

**AJAX implementation** Such a feature could be implemented in several ways. One approach would be to only reload the product category specific part of the search page depending on the product type the customer has specified. This could be implemented via Asynchronous JavaScript And XML or AJAX. The advantage of such a solution is that rather than reloading the whole page; only the product type specific parts are reloaded, making it faster.

**DHTML implementation** An similar approach would be to use DHTML. As of HTML 4.0 web page can be divided into several *div* elements as per specification. When a user first sees the advanced search page, only the *div* asking them enter information relevant to all product types is visible. The *divs* relevant to specific product types are invisible. The visibility of these can then be set via JavaScript in accordance with the product type selection the user has made.



## Chapter 4

# Conclusion and Outlook

### 4.1 Summary of Findings

Having compared *eSarine* to various commercial webshops and theoretical “best practice” models, I would recommend that the *eSarine* search function be modified to provide a user friendly interface, a ranking function and tolerance for spelling mistakes.

#### 4.1.1 Results Ranking

Because users only tend to look at the first few search results, products should be ranked in order of their relevance to a particular query. I demonstrated how this can be done by using the vector space model. Because of the nature of electronic commerce sites I suggest that query and document key terms, once identified don’t need to be weighed other than perhaps a within document frequency weight.

#### 4.1.2 Correcting Spelling Mistakes

Users frequently make spelling mistakes and a search engine should therefore be able to handle them robustly. For performance reasons it is best to split search through the dictionary into two parts. First a rough search should be used to identify possible correct spellings of the word in question. A finer search should then be used to reduce this list even further. After taking *eSarine*’s special needs into account, particularly its need to work in several languages, I would suggest using n-grams count for the rough search and Levenshtein-Damerau distance for the fine search. Both of these techniques would offer better results than phonetic matching algorithms which by their nature are restricted to one language.

If part of a user's query does not exist in the dictionary, alternative spelling suggestions should be displayed on the same page as search results rather than being corrected transparently. This way users searching for rare terms will not be shown false results.

### 4.1.3 The Search Interface

In order to be usable, the *eSarine* needs an intuitive search function that works like other search engine that users are used to. Because most users only use the simple search box, focus should be placed on improving it, even if this is at the expense of the advanced search page. Most importantly, the search box should be modified to accept any input, not just product titles, thereby bringing into line with all other popular search engines. A further improvement would be to the user could be allow the user to restrict search to a single category like on amazon.com.

The advanced search page's most useful feature, namely allowing a user to search product type specific fields is currently very well hidden. In order to make this feature easier to use, I would suggest implementing an intuitive tab based interface.

## 4.2 Outlook

This project has been entirely theoretical in nature. The next step would be to implement them and produce a working prototype. This is the only way to test some of the principles lay out in this report as well as being the only way to answer unanswered questions.

I personally would be very interested to know the answers to the following questions:

- Is the interface I suggest really significantly better than the current one?
- Can the interface be improved even further?
- Is the vector space model efficient?
- Do unweighed key terms produce acceptable results?
- If not, which weighing strategy is best?
- Does the spelling checker I suggest produce acceptable results?

The only way to know these answers is to program the new search engine and then test it on new users.

# Appendix A

## The Metaphone algorithm

A word is converted into its Metaphone key as follows [25]:

1. Apply the following to initial letters:
  - 'kn-', 'gn-', 'pm-', 'ae-', 'wr-' → (deleted)
  - 'x' → S
  - 'wh-' → W
  - retain all vowels
2. Remove all vowels in the rest of the word
3. Convert the remaining letters according to the following rules:
  - B → B unless at the end of a word after 'm as in 'dumb'
  - C → X (sh) if '-cia-' or '-ch-'  
C → S if '-ci-', '-ce-', or '-cy-'  
C → silent if '-sci-', '-sce-', or '-scy-'  
C → K otherwise, including '-sch-'
  - D → J if in '-dge-', '-dgy-', '-dgi'  
D → T otherwise
  - F → F
  - G → silent if in '-gh-' and not at end or before a vowel  
G → silent in '-gn' or '-gned'  
G → silent in '-dge-', etc., as in above rule  
G → J if before 'i', or 'e', or 'y', if not double 'gg'  
G → K otherwise
  - H → silent if after vowel and no vowel follows  
H → H otherwise

- J → J
  - K → silent if after 'c'  
K → K otherwise
  - L → L
  - M → M
  - N → N
  - P → F if before 'h'  
P → P otherwise
  - Q → K
  - R → R
  - S → X (sh) if before 'h' or in '-sio-' or '-sia-'  
S → S otherwise
  - T → X (sh) if '-tia-' or '-tio-'  
T → 0 (th) if before 'h'  
T → silent if in 'tch-'  
T → T otherwise
  - V → F
  - W → silent if not followed by a vowel  
W → W if followed by a vowel
  - X → KS
  - Y → silent if not followed by a vowel  
Y → Y if followed by a vowel
  - Z → S
4. Eliminate any adjacent repetitions of code
  5. If the resulting encoding is longer than four characters, truncate the code to four characters.

# Bibliography

- [1] T. Bray. On search. <http://www.tbray.org/> (Visited:12/06/2006).
- [2] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.
- [3] E. Chisholm and T. Kolda. New term weighting formulas for the vector space method in information retrieval. Technical report, Oak Ridge National Laboratory, Oak Ridge, Tennessee, March 1999.
- [4] H. Stormer & N. Werro D. Frauchiger, A. Meier. Zur entwicklung des struts-basierten webshops esarine. *HMD - Praxis der Wirtschaftsinformatik*, (238), 2004.
- [5] F. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, March 1964.
- [6] J. Zobel & P. Dart. A vector space model for automatic indexing. *Software - Practice and Experience*, 25(3):331–345, March 1995.
- [7] J. Zobel & P. Dart. Phonetic string matching: Lessons from information retrieval. In H. P. Frei, D. Harman, P. Schäuble, and R. Wilkinson, editors, *Proceedings of the ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 166–173, Zurich, Switzerland, 1996.
- [8] The European e Business Watch. Search users stop at page three. <http://www.ebusiness-watch.org> (Visited: 19/05/2006).
- [9] & C. S. Yang G. Salton, A. Wong. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, November 1975.
- [10] G. Salton & M. Gill. *Introduction to modern Information Retrieval*. Addison Wesley Publishing Company, 1988.
- [11] P. Graham. A plan for spam. <http://www.paulgraham.com/spam.html> (Visited: 19/07/2006).
- [12] P. Orfal & D. Harkey. *Client/Server Programming with Java and CORBA*. Wiley computer Publishing, 1 edition, 1998.
- [13] H. Hyyro. *Practical Methods for Approximate String*. PhD thesis, Department of Computer Science, University of Tampere, 2003.

- [14] University of Fribourg Information Systems group. Computer market studies (cms). <http://diuf.unifr.ch/is/research/itee/> (Visited: 01/06/2006).
- [15] V. Raghavan & G. Jung. A critical investigation of recall and precision values. *ACM transactions on Information Systems*, 7(3):205–229, July 1989.
- [16] D. Knuth. *The art of Computer Programming*, volume 2 Sorting and Searching. Addison Wesley Publishing Company, 2 edition, 1973.
- [17] S. Krug. *Don't make me think*. New Riders Press, 2 edition, 2000.
- [18] A. Meier. *Introduction pratique aux bases de données relationnelles*. Springer, 2 edition, 2002.
- [19] D. Mengmeng. Approximate name matching. Master's thesis, School of Computer Science and Engineering, Royal Institute of Technology, Stockholm, 2005.
- [20] L. Rosenfeld & P. Morville. *Information System Architecture for the World Wide Web*. O'reilly and Associates, 2 edition, 2002.
- [21] D. Frauchiger & A. Meier N. Werro, H. Stormer. esarine - a struts-based webshop for small and medium-sized enterprises. In *Proceedings of the EMISA Conference - Information Systems in E-Business and E-Government*, Luxembourg, 2004.
- [22] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.
- [23] BBC news. Search users stop at page three. <http://news.bbc.co.uk/2/hi/technology/4900742.stm> (Visited: 06/05/2006).
- [24] L. Philips. The double metaphone search algorithm. *C/C++ users journal*, June 2000.
- [25] A. Randall and B. Lait. An assessment of name matching algorithms. Available at <http://www.cs.utah.edu/contest/2005/NameMatching.pdf> (Visited: 09/06/2006).
- [26] G. Salton. *Automatic Text Processing*. Addison Wesley Publishing Company, 1988.
- [27] D. Lee H. Chuang & K. Seamons. Document ranking and the vector-space model. *IEEE Software*, 14(2):67–75, 1997.
- [28] J. Spolski. *User Interface Design for Programmers*. Apress, 1 edition, 2001.
- [29] A. Meier & H. Stormer. *eBusiness & eCommerce*. Springer, 1 edition, 2005.
- [30] C. J. van Rijsbergen. *Information Retrieval*. Butterworths, 2 edition, 1979. Online version available at <http://www.dcs.gla.ac.uk/Keith/Preface.html> (Visited:05/03/2006).
- [31] R. Lowrance & R. Wagner. An extension of the string-to-string correction problem. *Journal of the ACM*, 22(2):177–183, April 1975.