

ALTERNATIVE ARCHITECTURES FOR IMPROVING DOCUMENT
READABILITY

by

ANKUR BORA

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2006

Copyright © by Ankur Bora 2006

All Rights Reserved

ACKNOWLEDGEMENTS

I would like to thank Mr. David Levine for giving me an opportunity to work on this exciting project. Mr. Levine's support, guidance, advice and encouragement have been invaluable in the completion of this work. I would also like to thank Mr. Gil Carrick for guiding me through all stages of the thesis work. I acknowledge Mr. Carrick for being constantly accessible and supportive, for pointing me in all the right directions, and reading draft after draft of this thesis. I sincerely thank Dr. Manfred Huber and Dr. Gergely Zaruba for serving on my committee. I also offer my sincere gratitude to my parents Mr. Rabindra Chandra Bora and Ms. Maloti Bora for constantly encouraging me to obtain a higher educational degree in an institution of excellence.

April 10, 2006

ABSTRACT

ALTERNATIVE ARCHITECTURES FOR IMPROVING DOCUMENT
READABILITY

Publication No. _____

Ankur Bora, M.S.

The University of Texas at Arlington, 2006

Supervising Professor: David Levine

A number of tools have been developed to improve the readability of documents. These tools assist users to make changes that result in an easy to read document. Most of these tools were developed for users who work independently with little interaction with other users. However, in recent years, because of the proliferation of the Internet, there has been increased collaboration between users. A user may need to share his document repositories with others so that both can benefit. Another set of users may work with information from a specific domain. These domains may be

located in different geographical areas. Some users may like to import the result of the readability tool to other applications.

With improved computer processing power, storage capacity and transmission speed it is possible to develop systems to meet such needs. Current development in Graphical User Interfaces has also made it possible to develop a better editing tool. With these tools, it is possible to offer users more alternatives to restructure and simplify the document. This thesis describes a number of techniques to develop tools for improving document readability both in local and distributed environments. A Prototype has been developed for each of these techniques and it has been found that such systems are feasible and practical.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS.....	iii
ABSTRACT	iv
LIST OF ILLUSTRATIONS.....	ix
LIST OF TABLES	xi
Chapter	
1. INTRODUCTION.....	1
1.1 Classifying the users.....	2
1.2 Data restructuring with human involvement.....	5
1.3 Objective of this thesis.....	6
2. BACKGROUND	7
2.1 Document Readability.....	7
2.2 Data Accumulation	9
2.2.1 Accumulating the Lexicon.....	10
2.2.2 Accumulating the corpus.....	12
2.3 Motivation.....	12
3. METHODS AND TECHNIQUES.....	15
3.1 Developing the Process	16
3.1.1 Calculating the readability score.....	16
3.1.2 Developing the Parser.....	21

3.2 Designing the Windows stand-alone application.....	25
3.2.1 GUI for Statistical and Readability calculation	27
3.2.2 GUI for editing and improving words.....	28
3.2.3 GUI for restructuring sentences	29
3.2.4 GUI for updating sentence.....	30
3.2.5 GUI displaying modified sentence.....	31
3.2.6 GUI for modifying phrases.....	32
3.2.7 GUI for Report.....	33
3.2.8 GUI for user feedback	34
3.3 Designing the Client Server application.....	35
3.3.1 Organization of Web Services	36
3.3.2 Logical flow of data in Client Server architecture.....	38
3.3.3 Client in Client Server application.....	39
3.3.4 Server in Client Server application	42
3.4 Designing the distributed architecture using Software Agent.....	46
3.4.1 Developing the Software Agent.....	47
3.4.2 Architecture of the Software Agent.....	48
3.4.3 XML for Software Agent.....	51
3.4.4 Logical flow of data in Software Agent	52
3.4.5 Updating lexicon with Remote Agent	54
3.5 Designing the Internet based architecture	54
3.5.1 Logical flow of Data in a Web Application.....	55

3.5.2 Client Script processing.....	60
3.5.3 Web Interface of IDR application	61
4. CONCLUSIONS AND FUTURE WORK	63
4.1 Conclusions.....	63
4.2 Future Work.....	64
Appendix	
A. EXPERIMENTAL RESULT.....	66
B. READABILITY FIGURES FOR SAMPLE DOCUMENTS	71
C. READABILITY STATISTICS COMPUTED FOR DOCUMENTS.....	74
REFERENCES	78
BIOGRAPHICAL INFORMATION	80

LIST OF ILLUSTRATIONS

Figure	Page
1.1 Different users using different Lexicons	4
3.1 Objects defined in our System.	19
3.2 Parser translates input text into a number of objects.....	21
3.3 A text consist of Documents, Sentences and Words.....	22
3.4 Physical flow of data in Windows stand-alone environment..	25
3.5 GUI for Statistical and Readability calculation	27
3.6 GUI for editing and improving words.....	28
3.7 GUI for restructuring sentences	29
3.8 GUI for updating sentence.....	30
3.9 GUI displaying modified sentence.....	31
3.10 GUI for modifying phrases.....	32
3.11 GUI for Report.....	33
3.12 GUI for user feedback.....	34
3.13 Physical flow of data in Client Server Environment.....	35
3.14 Web services hosted by Web Server	36
3.15 Logical flow in Client server	38
3.16 Client in Client Server.....	39
3.17 Server in Client Server	43

3.18	Data flow in Software Agent solution.....	48
3.19	Lexicon data in XML format.....	51
3.20	Logical flow of data in Software Agent	53
3.21	Updating lexicon with Remote Agent.....	54
3.22	Logical flow of Data in a Web Application	56
3.23	Data flow in Web vs. Distributed environment	57
3.24	Client Script processing.....	60
3.25	Web Interface of IDR application.....	62

LIST OF TABLES

Table	Page
2.1 A Typical lexicon entry	11
2.2 Some of the test files	12
3.1 Comparing Web Services with other standards.....	44
3.2 Actions performed by Client Script.....	61

CHAPTER 1

INTRODUCTION

Improving document readability can be defined as a process that reduces the reading complexity of a text while attempting to preserve its meaning and information content. The aim of document simplification is to make text easier to comprehend for a user or processing by a program.

Readability metrics are intended to measure the difficulty of understanding a passage of text. The difficulty is measured in terms of the word and sentence complexity [Samuels Zakaluk 88]. Each of these metrics defines a score to evaluate reading ease. However it has been observed that these surface attributes of a text alone are not sufficient to reduce the difficulty of a text [Si Callan 05].

A difficult document becomes easier to read if some of the complex words are replaced with easier ones. This is a basis of document readability automation, to offer a user with a list of simpler words to replace the complex words. This collection of words is critical for the success of improving document readability application. A wider collection provides more flexibility to the user.

Another important aspect is the requirement and need of specific sets of users. Certain users may be interested in specific area of specialization-the legal field for example. A readability improvement application should provide this user with an up-to-date collection of words from legal field. It is also possible that a group of users will try

to create a centralized repository of words exclusively for that group. The application should also provide that functionality.

With the advent of Internet based distributed architectures it is possible to implement this functionality in document readability automation. Extensible Markup Language (XML) can be used for message passing between applications. The Document Object Model (DOM) provides a rich set of functionality to retrieve data from XML efficiently. Simple Object Access Protocol (SOAP) can also be used as a network transport mechanism to transfer data efficiently. The Web Service is also an efficient technique to connect to the heterogeneous data source and retrieve information in an efficient manner.

These new developments in technology have made it possible to implement all the requirements and needs of single user or group of users. In this thesis, all these different techniques to improve document reading will be discussed. We will use the term IDR (Improving Document Readability) to refer to the window tool that we have developed.

1.1 Classifying the users

We have divided the users (of the readability improvement application) into three groups:

Regular user: These are conventional users who use the main lexicon maintained in a central server. A lexicon is a list of words with related information. We explain this term in detail in Chapter 2.2. A portion of this main lexicon is copied to the

regular user's machine at the time of installation of the application. Usually this portion of lexicon is not updated frequently although there may be periodic modification. A regular user may also have a private lexicon to keep his name, contacts, acronyms etc.

Corporate user: These are a group of users sharing a specific lexicon. Apart from the main lexicon, there is corporate lexicon for these sets of users. For example a user who is from the medical profession is more likely to use medical terms. As such, a separate medical lexicon can be maintained for these users. A medical professional can contribute to this lexicon, which will benefit other users. Other examples of corporate users are people from the educational and legal fields. These groups of users may be working on a certain project and they may need to maintain and share documents related specifically to that project. Normally a corporate lexicon is going to be in the evolving phase for a longer duration. The professional users will continue to contribute to the corporate lexicon. Our system has to provide the mechanism to incorporate these changes.

Web user: In a web application, the entire application runs in a Web browser. A Web application uses Internet technology to integrate desktop, work group and enterprise computing into a single cohesive framework. Anyone using this framework can be a Web user. However, because of the diversity of users and their computing environments, the architectural complexity also increases in Web applications.

The scope of readability improvement applications increases manifold if these requirements are supported. For example, the application may have to provide data

exchange between corporate users. There may be a need to provide a mechanism for notification of events so that remote user will know about any update on central server.

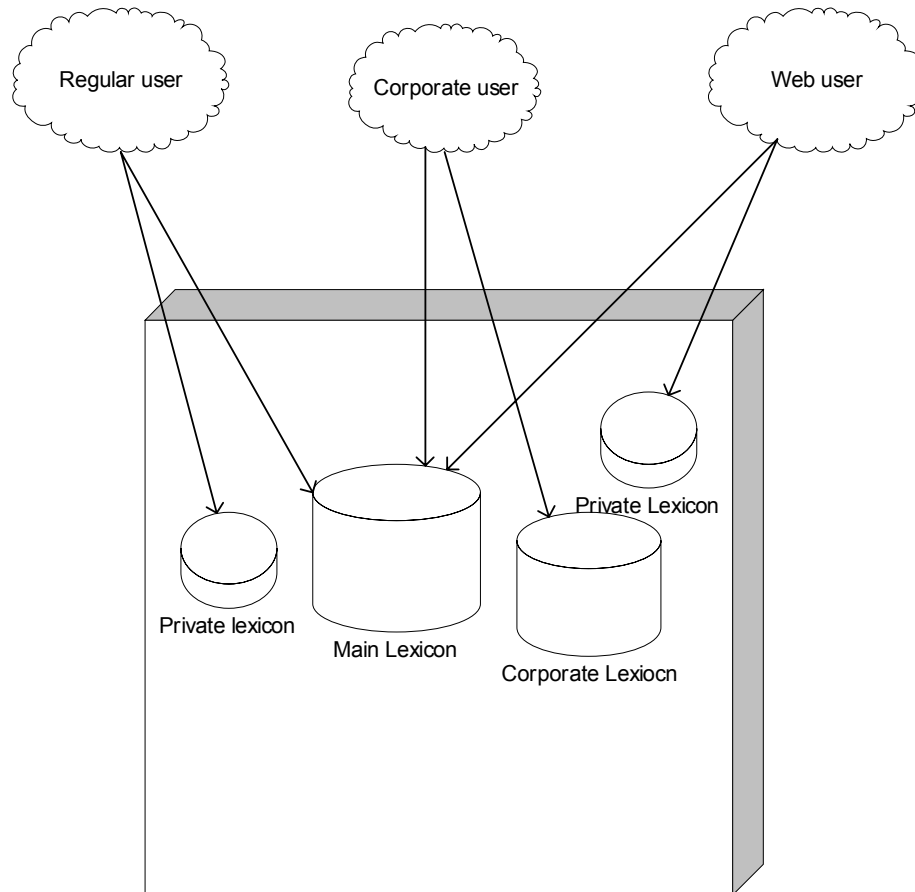


Figure 1.1 Different users using different Lexicons

As shown in the figure 1.1, a user may use the main lexicon, the corporate lexicon or can have a private lexicon. Before deciding the implementation we have to consider the user's needs and requirements. The implementation for the corporate user may be different from the regular user. It is therefore necessary to identify the right implementation strategy considering the requirement of the users involved.

Our approach is to give priority to user requirements and design the system based on these requirements. Accordingly we have developed four separate implementations to cater to these diverse groups of users:

- Stand-alone environment – This window based process is developed for regular users.
- Distributed environment using Client Server – This process is to cater to corporate users.
- Distributed environment using Software Agent – This process is to facilitate the requirements of corporate users.
- Internet based architecture – This process is developed for Web based users.

1.2 Data restructuring with human involvement

Readability formulae use a number of linguistic variables to predict difficulty of a document. Their underlying assumption is that improvement in document readability can be achieved by reducing word and sentence complexity. However these formulae alone can't always lead to a better readable document. Our experience is that with the current state of the art, the simplification is possible only with human involvement. We feel that a semiautomatic interactive tool can be developed to facilitate human interaction for a readable document.

1.3 Objectives of this thesis

The Current state of the art technology, the World Wide Web (WWW) and the Internet have facilitated increased collaboration between groups of users. User from one corner of the world can easily communicate with user in another corner. A user can run an application located in a distant server through a Web browser effortlessly. These new developments have significantly changed the scope of document readability. It is possible that a document readability tool can be developed to support the user in stand-alone, distributed and Web environments. This thesis proposes a number of techniques to develop an improving document readability tool. We will discuss these alternative architectures and whether we are able to meet our objectives.

CHAPTER 2

BACKGROUND

Documentation is a critical element of an organization. Poor presentation can result in lower reader acceptance of written material [Klare 77]. There is evidence that the easier technical material to read, the more likely it is to be used. Conversely, if the technical document is difficult relative to the reading level of the intended audience, it is more likely that a user will seek alternative methods to secure necessary information [Klare 55] Readability formulae were developed initially for educational documents. However, in the course of time, users started using them for other areas like technical documents, course material of a correspondence course, computer language manuals, major computing periodicals, contractual, legal or regulatory government documents etc.

2.1 Document Readability

A text is considered readable if the intended readers are able to read it quickly, accept it (i.e., persevere in reading it), and understand it clearly [Klare 77]. An information system manager might be interested in the availability of simple, easily administered, objective methods that predict potential reader difficulty with written materials. Such method would be useful, for instance, in assessing internally generated text serving as communication links among systems analysts, programmers, operators,

end users, auditors and managers, or documentation supplied by hardware and software vendors [Guillemette 01]. It is also observed that in some other areas like educational correspondence courses, dissatisfaction with poor technical writing has adversely affected user behavior. For instance, [Klare Smart 73] found a significant relationship between the ratio of students completing a correspondence course and the use of more understandable course materials.

Readability formulae have been developed to estimate the potential difficulties reader might have in understanding a document. The process of computing these formulae is time consuming because it involves counting of text. However, with advancement in information technology, readability formulae have been automated. This has significantly improved rapid assessment of text samples.

Readability formulae use a number of linguistic variables to predict the difficulty of a document. For example, one of the indicators of comprehension difficulty is vocabulary load [Dale and Chall 48]. The three most common methods to measure vocabulary load are: rarity or difficulty of words, diversity of words, and word length. Sentence structure also provides a significant indicator of comprehension difficulty. The most common method of approximating sentence difficulty is by calculating average sentence length. This metric has been seen to correlate highly with other measures of grammatical complexity [Bormuth 66].

There are a number of readability formulae developed for users of different ages and groups. Many of these formulae are designed and validated for some small-specialized category of documents such as primary grade levels or technical

publications. Several formulae, however, have achieved broad acceptance [Carrick 2000]. Some are more formally defined but difficult to calculate manually. Others are simple enough to do by hand and have thus gained a wide audience. Among them, the Flesch [Flesch 48] Reading Ease formula has often been used to examine the readability of technical documents. Gunning's [Gunning 52] Fog Index is mostly used in business and government. Dale-Chall [Dale and Chall 48] formula is considered as most accurate. For our implementation, we are using the Flesch Reading ease. This is selected because it is popular and therefore would have more sample documents.

2.2 Data Accumulation

According to our objective, we are developing a system that will provide the user better alternatives for document restructuring. Since a word is a basic unit of a document the system should contain the relevant information of a word. This relevant information can be the syllable count or the synonyms of the word. We call this set of information the lexicon. It is also important that we collect enough documents to test our application for the widely diverse need of users. We call this a corpus. A corpus is a collection of text for computer analysis. We have used extensively the data repository collected by Albert Gilmore Carrick [Carrick 90]. The lexicon has been accumulated over a period of time spreading years. The data from public lexicons namely Moby Lexicon [Ward 96] and the Shorter Oxford English Dictionary [OxUP63] have been used to develop this lexicon.

2.2.1 Accumulating the Lexicon

The lexicon is consisting of rows of words, each word in turn includes a set of associated properties represented by the column of that row. A lexicon in our vocabulary includes the following fields or properties:

- Word - Currently our lexicon contains over 150 thousand actual words.
- Capitalization flag – In our lexicon the word is always stored in lower case. This flag indicates if it is a capitalized word. A value of ‘C’ in the capitalization flag of a word indicates that it is a capitalized word. An example is the word “Adam”. This word is always capitalized so the flag is ‘C’. A “c” indicates that the word may be either capitalized or not. An example word is “god”; this word can either be “God” or “god”. An “A” indicates that the word is always written in all caps. An example is “ACM” (Association of Computing Machine).
- Syllable count - The number of syllables in the word.
- Flag Byte - Flag bytes are indicators that can be used for editing words. For example, the word “goodbye” should be represented as “good-bye”. So, the Flag byte for the word “goodbye” is “H” indicating hyphenation. Similarly “A” indicates that this word is an abbreviation. For example, the word “jr.” is an abbreviation and it has a flag of “A”.
- Synonym list -This field contains the synonyms of a word. The synonyms are different words with similar meanings and are interchangeable.

Table 2.1 A typical lexicon entry

Word	Flag	Syllable count	Capitalization Flag	Synonyms List
dr		3		doc, MD, M.D.
builtin	H	2		Built-in
cpu		3	A	
gvt	A			Government
carpet		2		rug, mat
rug		1		
mat		1		
mastoidectomy		5		procedure, operation, surgery, treatment

Table 2.1 is a portion of the lexicon displaying the fields of a word. Some of the entries are explained as follows.

The entry for the word “cpu” has a Capitalization flag of “A” which indicates that this word should always be displayed as CPU. Knowledge about required capitalization is used in determining sentence endings.

The word “Builtin” has a Flag “H” indicating that this is word should be hyphenated.

The word “dr” has a Flag “A” indicating that this word is abbreviation for the word doctor.

The word “carpet” has a Syllable Count of two and it has synonyms “rug” and “mat”. Both the words “rug” and “mat” have Syllable count of one.

Similarly the word “mastoidectomy” has a Syllable Count of five and it has synonyms “procedure”, “operation”, “surgery” or “treatment”.

2.2.2 Accumulating the corpus

We have accumulated several text files from different sources. These text files constitute the corpus of text for testing our Improving Document Readability application. These files cover a wide range of topics including novels, government reports, children's book, historical documents and science related topics. Many of these samples are from Project Gutenberg [Hart 99]. A few are part of the Brown University corpus [FrKu 79]. Other samples are from the literature about readability formulae. They are included so that we can validate our implementation of the formulae.

Table 2.2 Some of the test files

Text file name	File size in Bytes
Ad Copy	3316
Bureau	2611
Child	1513
Conventional	706
Easy Writer	581
Example	1384
Fog296	327
Fig88	231
Fog9352	182
GMK3	2463
GoodGov	3657
GovRep	2791

2.3 Motivation

A number of readability tools have been developed based on the readability metrics. They primarily compute and display the readability score. A few also suggest

shorter words to replace the existing complicated words. It has been observed that most of the time this approach is not efficient. Because the words were not shown to the user in context, the edited sentences were often garbled [Carrick 2000]. In many cases, the sentence became unreadable after these words were replaced. This happens because a sequence of words has a meaning that would not be apparent from considering the individual words alone.

The observation made by [Carrick 2000] is that the editing needs to take place with the system showing the context of the words being edited. We are developing a Graphical User Based tool, which we call Improving Document Readability (IDR). This tool will provide the user with alternative words that are less complex than the original. We define the complexity of a word in terms of syllable count. The syllable counts of the alternative words suggested by the system are less than the syllable count of the original word. We are currently using Flesch Reading ease as the readability metrics. We also made provision in our application to support other readability metrics like Gunning's and Dale-Chall.

It is expected that group of users that we defined earlier will use the readability tool that we are developing. The users can be reading specialists, teachers, trainers, librarians, instructional writers, curriculum writers, policy and procedure writers, technical writers etc. Our tool can also be used by a group of users who can work together on specific subject. For example, there may be a group of users working on technical documents and these users are located in different geographical locations. The requirements of these users will be different from a user who works stand-alone. For

these groups of users, our system will have to provide a mechanism for the sharing of lexicon and exchanging private lexicon. We have proposed a Client Server architecture using Web Services and Client server using a Software Agent.

There can also be users who may like to use our readability application over the Web. For example, a teacher may like to know whether the course material that he is preparing for the examination is readable for his class standard. He can copy and paste the reading material in our Web application and find the readability score. He can also use the interactive feature of our tool to reduce the word and sentence complexity. We have proposed developing an Internet based Architecture for this type of user.

We have developed alternative architectures namely IDR for stand-alone, IDR for distributed environment using Client Server, IDR for distributed environment using the Software Agent and IDR for Web based application.

CHAPTER 3

METHODS AND TECHNIQUES

We have followed the following steps in developing our application:

- Prototype design – This step is to identify the user requirement and accumulate the data. For this we have studied different readability formulae and their implementation.
- The initial prototype – To construct an executable software model based on either an initial selection of functions or on user's needs that have been identified. In our case, we have developed versions for Windows stand-alone, distributed and Web environment.
- The prototype testing – Executable software is tested against a set of test cases. We have used the text files as mentioned in Table 1.2.
- The prototype evaluation – Based on the result of testing, the executable software is evaluated. If it meets the standard the prototype is accepted; otherwise the prototype is redesigned. For our implementation, we have developed a number of use cases to verify whether the executable has met the requirements.

While designing the Improving Document Readability tool, we followed these steps iteratively.

3.1 Developing the Process

This section first focuses on different readability formulas that have been developed. We discuss the one that we use in our application to calculate the readability score of a document. We also developed a parser. The objective of the parser is to parse a document and convert the document into collections of words and sentences. This way the system can readily recalculate the score whenever any modification is made to the document.

3.1.1 Calculating the readability score

A readability formula is a mathematical equation that is meant to predict the level of reading ability needed to understand a particular piece of prose. These equations consider a number of features of a document to calculate the reading ease of that document. Our system calculates the readability score based on existing readability formulae. This thesis assumes that merit of these formulae is established.

Readability formulae are intended to identify the difficulty of understanding a passage of text. The approach taken by these formulae is to calculate the reading ease of a document by a numerical score. This score can also be converted to an educational grade level.

The Flesch Reading Ease Scale is the most widely used formula outside of the educational field. It is an easy formula to use, and it makes adjustments for the higher end of the scale. It measures reading from 100 (for easy to read) to 0 (for very difficult

to read). The output of the Flesch Reading Ease formula is a number from 0 to 100, with a higher score indicating easier reading. This formula is calculated as follows:

$$206.835 - (1.015 \times \text{ASL}) - (84.6 \times \text{ASW})$$

Where:

ASL = average sentence length (the word count divided by sentence count)

ASW= average number of syllables per word (the syllable count divided by words counts)

As per this formula we observed that computing the readability score of a document depends on two factors:

- Average sentence length of the document.
- Average complexity of the words of that document.

Accordingly, to improve document readability, an application has to provide user the following functionality:

- Display the longer sentences so that the user can edit and make them shorter.
- Display the complex words with option to replace them with simpler words.

Our readability system has been developed to achieve this objective. The parser reads the input file and constructs a number of objects. Here we use the term object to indicate a self-contained entity having unique values and properties. For example, a “Word” object has a property called “Number of Syllables”; a “Sentence” object has a property called “Number of words”. The “Sequence” value of a “Word” can be five. This indicates that it is the fifth word in a sentence. Objects are the building blocks of our algorithm. The algorithm calculates the readability score based on the properties of

these objects and dynamically regenerates and recalculates the score based on the user's actions.

There is a co-relation between our objects and real world units. For example, our objects "Sentence" carry the same meaning as the sentence of a paragraph. However, real word units are not precisely defined. For example one user may consider a ":" as end of a sentence while for another user ";" may indicate the continuation of the same sentence. The algorithm defines these objects based on the following rules:

- A "Sentence" is defined as a string of words punctuated with a period (.), an exclamation point (!) a question mark (?) or a ";".
- A "Word" is consists of string of letters separated by space or comma or a period. Any character among ":}]!?">" is also considered as word separator.
- Hyphenated words are considered as one "Word".
- The object "Word" has a property called syllable count. This value is defined in the lexicon. Currently, in case, this value is not found in lexicon, a system defined default is assigned to the syllable count of that "Word". We will have a provision whereby the user can enter syllable count if it is not defined in lexicon.
- Sometimes numbers are represented in numeric form. In that case they should be interpreted to text to determine the syllable count. For example, if the word "15th" is encountered we should treat it as "fifteen" for calculating syllable count. This functionality has not been implemented currently.

- Abbreviated word should be read as unabbreviated to determine the syllable count of that “Word”.
- The object “Word” has a property called synonym list and this value is read from the lexicon.

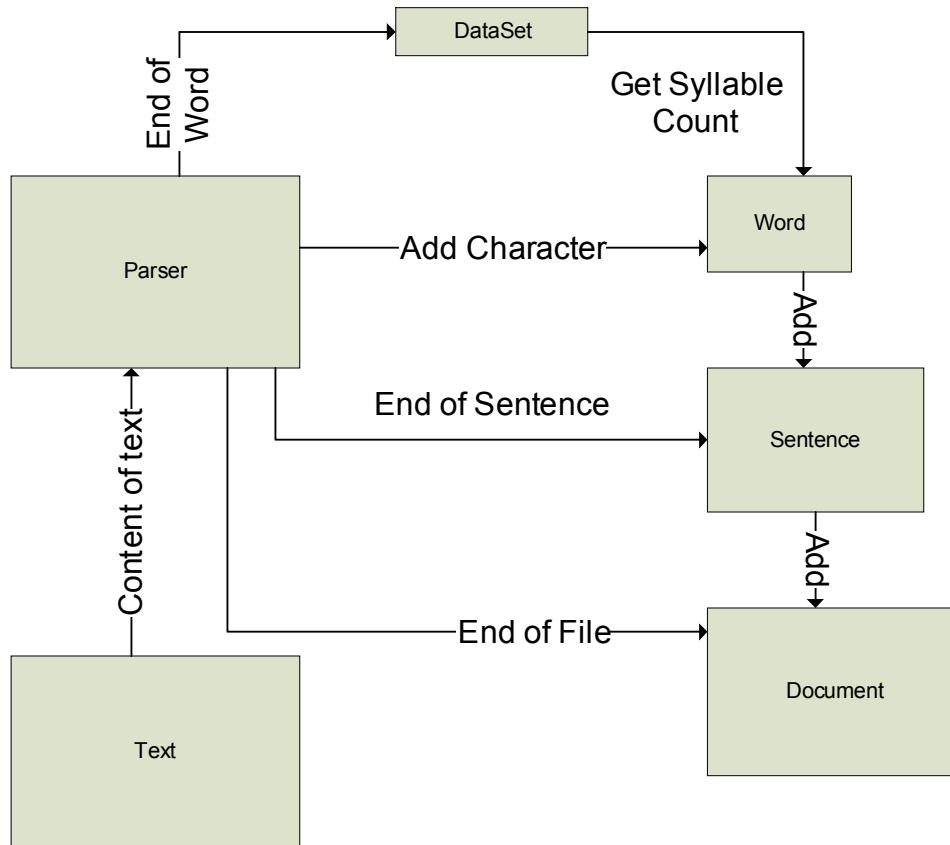


Figure 3.1 Objects defined in our System

We have developed our application using object oriented analysis and design. As shown in the figure 3.1 the Parser object reads the content of the Text object. As it continues parsing the text, the Word, Sentence and Document objects are instantiated.

The Parser object also interacts with the Dataset object to retrieve the syllable count for the Word object. These objects are explained as below:

Text: This object is responsible for reading the input file selected by the user.

Parser: This object parses the input from the Text object. It reads the text one character at a time and depending on the input character it executes different actions:

- If it encounters a character it notifies the Word object to add.
- If it encounters space and semicolon it indicates end of Word. As indicated above, there are several other characters that can end a word. For a word, we define the property named “Separator”. Separator contains the characters separating two words. It also retrieves the syllable count and other data from the Dataset object and adds them to the Word object.
- A period (.), an exclamation point (!) or a question mark (?) etc. indicates the end of the sentence and the Parser notifies this to the sentence. Similar to the word, a Sentence has a property “Separator”. The “Separator” contains the characters between two sentences.
- An EOF (End of file) indicates the end of the Parser operation.

Sentence: A sentence begins with a non White space character and ends with either a period (.), an exclamation point (!) a question mark (?) or a “;”. We define a property named White Space. A White Space is a string of space, tab, carriage return and other characters. When the Parser scans a text, it reads the text one character at a time. A non White space character indicates the beginning of a sentence. This also implies the beginning of a word. A word delimiter like space or semicolon indicates the end of a

word. The Parser will read the next word. Finally it encounters either a period (.), an exclamation point (!) a question mark (?) or a “;” which indicates end of the sentence.

3.1.2 Developing the Parser

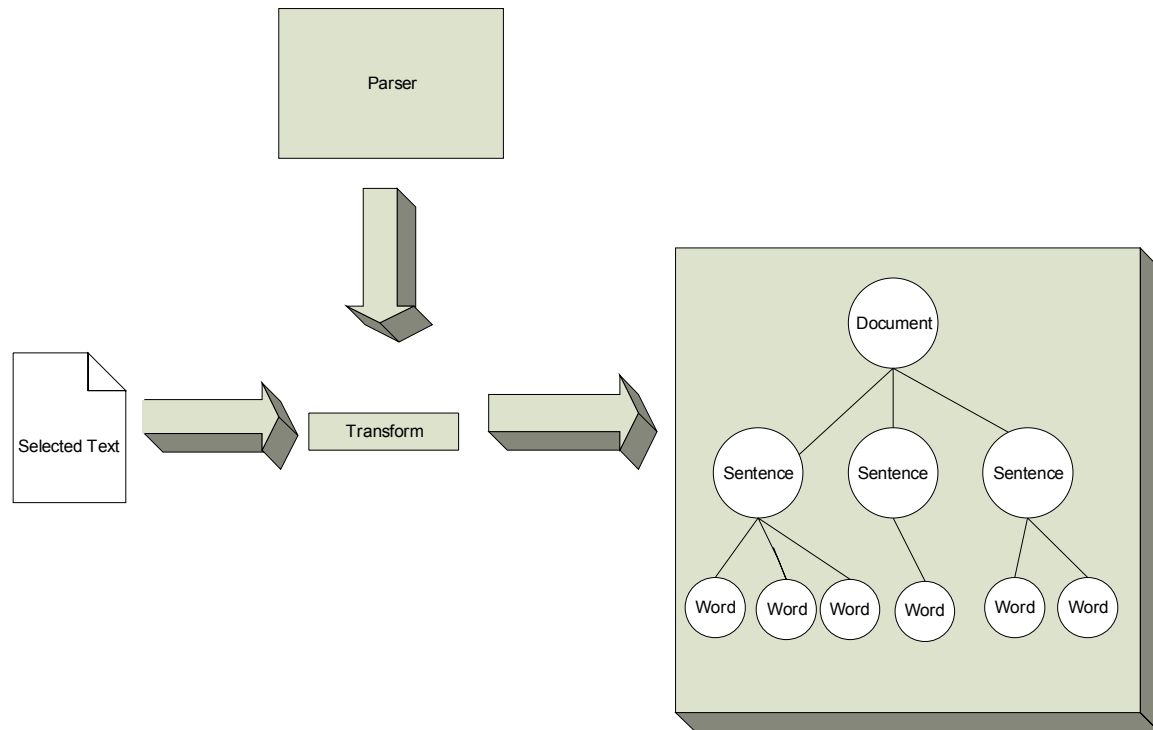


Figure 3.2 Parser translates input text into a number of objects

We have developed the Parser class to parse the input document selected by the user and also to collect the statistical information about the document. The Parser object maintains this information in memory till the users select another document or exit the system. A Parser object is instantiated when the user selects an input text file and starts the processing. The Parser parses this input file and creates a collection of word, sentence objects and one document object.

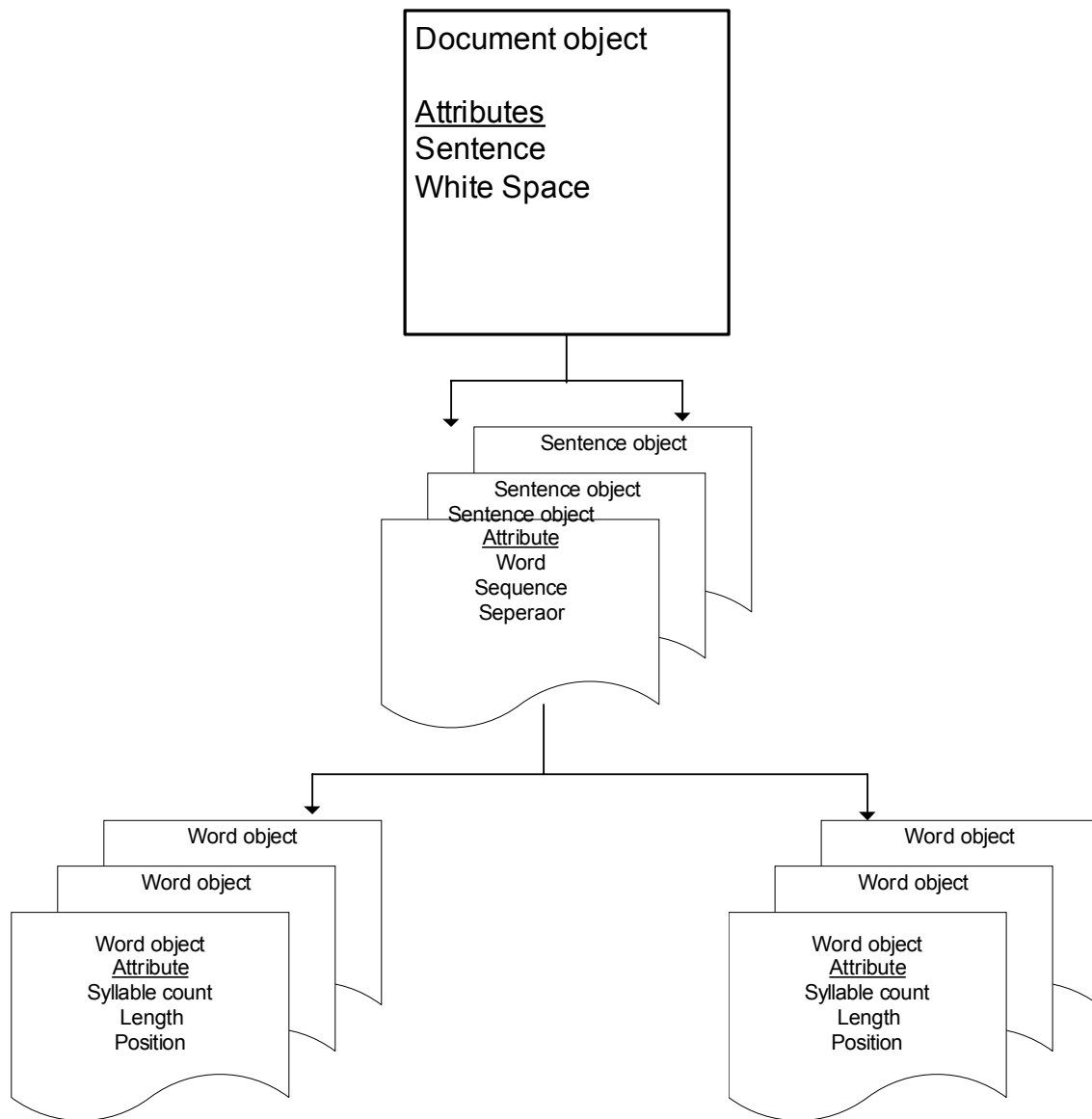


Figure 3.3 A text consist of Documents, Sentences and Words

Word: The Word is the basic building block of the document. A sentence consists of a number of words and words are separated from each other by a string of delimiters. We have defined a number of attributes for words. They are Word Length,

Position, Syllable, Flag and Separator. Word Length is the number of characters in a word; Position indicates the position of the word within the sentence; Syllable is the number of syllables in that word. Flag indicates whether the Word is a special object like an abbreviation. Separator is an attribute which maintains the string of delimiters that separate one word from another. These attributes of the word are populated when the parser scans the document.

Sentence: A Sentence is a collection of words. Normally a sentence is separated from another sentence through a period, an exclamation point (!) or a question mark (?). The attributes of the sentence are:

- Number of words – The application uses this attribute to determine the longest sentence, the second longest and so on. This attribute can also be used to sort the document on sentence length.
- Separator – Two sentences are separated by a string of characters. They can be a string of space, a Carriage Return or a Tab. The attribute Separator keeps this information for each sentence.
- Sequence – This attribute maintains the position of the sentence within the document. This attribute can be used to address a particular sentence, for example the third sentence in the document.

Document: A Document consists of a collection of Sentences. It is possible that a document may contain rows of empty space before the first sentence begins. We have defined an attribute named White Space to keep this information.

We are developing our IDR application based on the objects that we defined. The prototype of IDR is developed for a Windows stand-alone environment, a distributed environment using Client Server, a distributed environment using the Software Agent and an Internet based architecture. The IDR tool provides the following functionality to improve the readability score of a document.

- Words simplification: The IDR tool identifies the complex words and provides simpler alternative words.
- Sentence simplification: The IDR tool identifies the longest sentences and allows the user to break and edit them into smaller ones.
- Phrase simplification: This option of IDR allows user to remove unnecessary words or phrases of a sentence.

It is expected that a user can use this functionality of IDR to improve the readability score of a document.

We have a common Graphical User Interface for all these four environments. We have developed the graphical form initially for the Windows stand-alone application. This form is reused or extended for other environments.

3.2 Designing the Windows stand-alone application

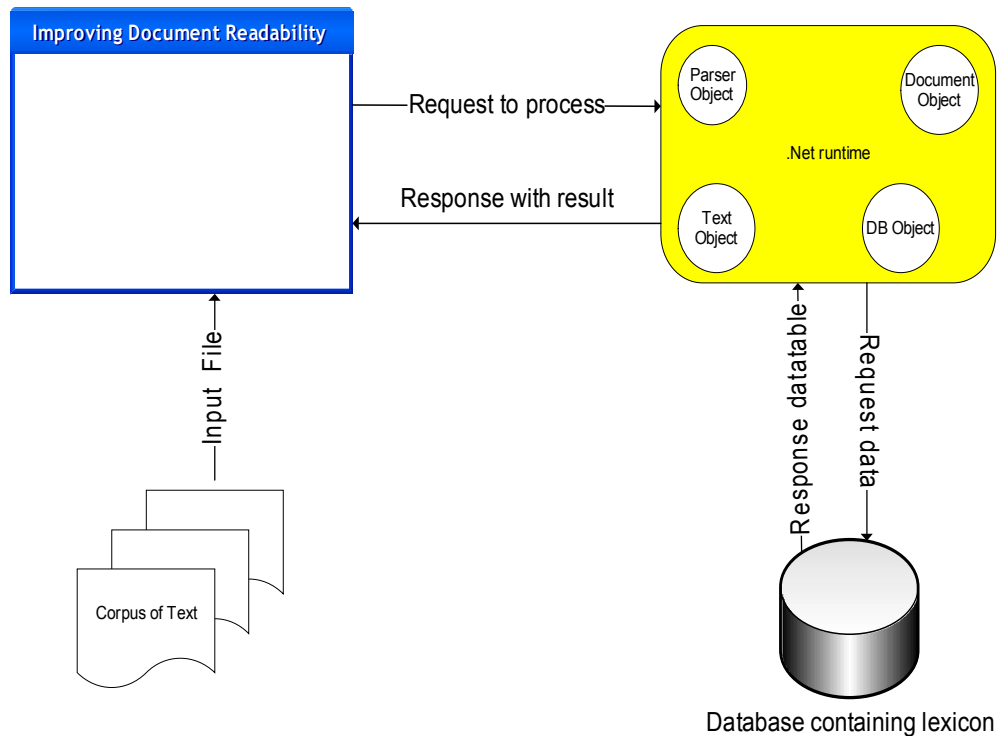


Figure 3.4 Physical flow of data in Windows stand-alone environment

In a Windows stand-alone application, all the modules of the application reside in the client machine. This application is suitable for a regular user who does not need a frequently updated lexicon. Normally, the lexicon is set up along with the application at the time of installation. As shown in the figure, the Corpus of Text and the database containing the lexicon are physically located in the user's machine. When a user runs the application, the components are instantiated. While Parser object is responsible for parsing the text, the DB object is responsible for connecting to the database and accessing the lexicon data. The application will need the lexicon frequently as the user continues editing the document. A frequent request/response from the database will

slow down the editor considerably. To prevent this, as soon as the application is loaded, the DB object is instantiated, and it connects and retrieves the lexicon in a data table. The Application can access the lexicon as a table in memory. Accessing data from primary memory is faster than accessing from secondary memory. Since the lexicon remains in primary memory throughout, we get a better response time.

We next discuss the functionality provided by Our IDR tool. The IDR tool allows user to select a text file and view the readability score of that text document. The GUI also displays those words that are considered as complex. The complexity is measured in terms of syllable count. A user can edit the document and save it once the editing is over. The IDR tool also displays the longer sentences of a document and provides the option to edit them. A user can edit and break the sentence into shorter sentence. According to The Flesch Reading Ease formula, the reading complexity of a text increases if the document contains longer sentences. A User can use our tool to determine the longer sentences and can reduce the reading complexity of the text. The IDR tool also provides an option called “Report” to view the statistics of the document. The Report option displays the number of words, number of sentences, and syllable count of the document. These values are dynamically generated, as the user continues editing the documents, IDR recalculates these values. This way a user can determine if his editing is reducing the reading complexity of the text.

3.2.1 GUI for Statistical and Readability calculation

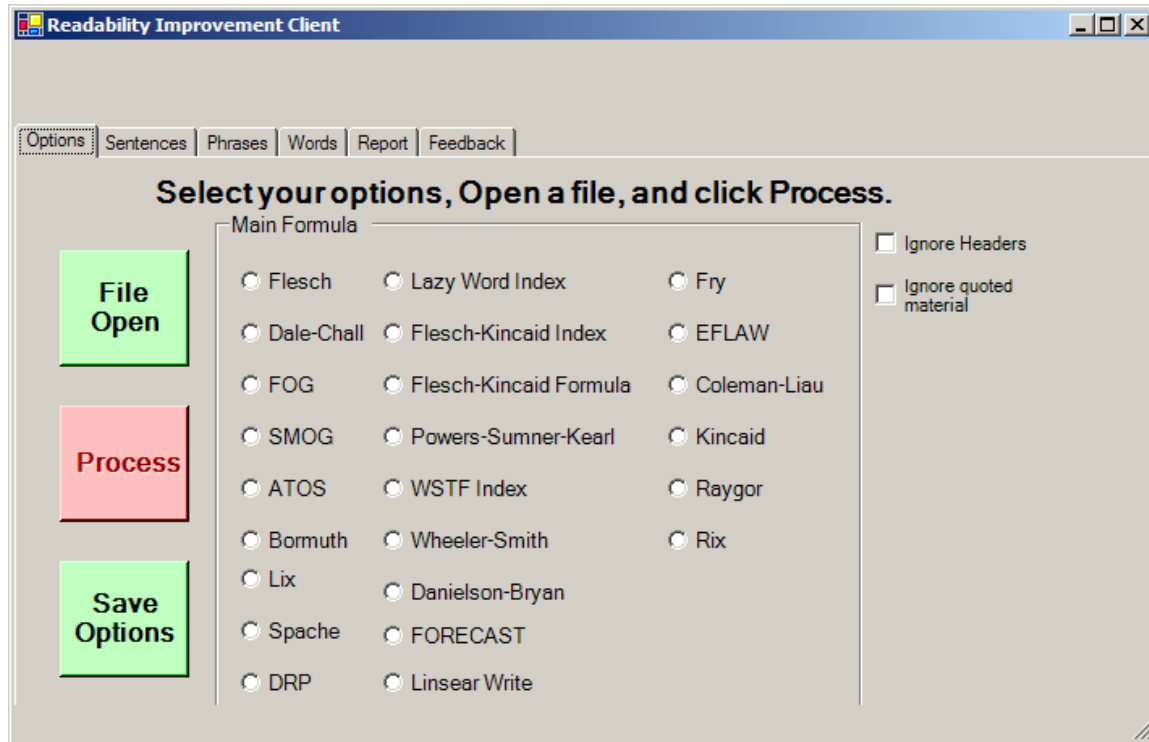


Figure 3.5 GUI for Statistical and Readability calculation

As shown in the figure 3.5, the Options tab allows the user to open a text file for processing. When the user clicks on the File Open button, a Windows file open dialog box appears. The user can select the intended file. When the user clicks the Process button, the IDR tool starts processing the file. The file is processed and readability statistics are calculated. The user can click on the Report tab to view these statistics once processing is complete. As shown in the figure 3.5, the user can select any of the Readability formulae to measure the reading difficulty of the document. Currently we are implementing the Flesch Reading Ease formula.

3.2.2 GUI for editing and improving words

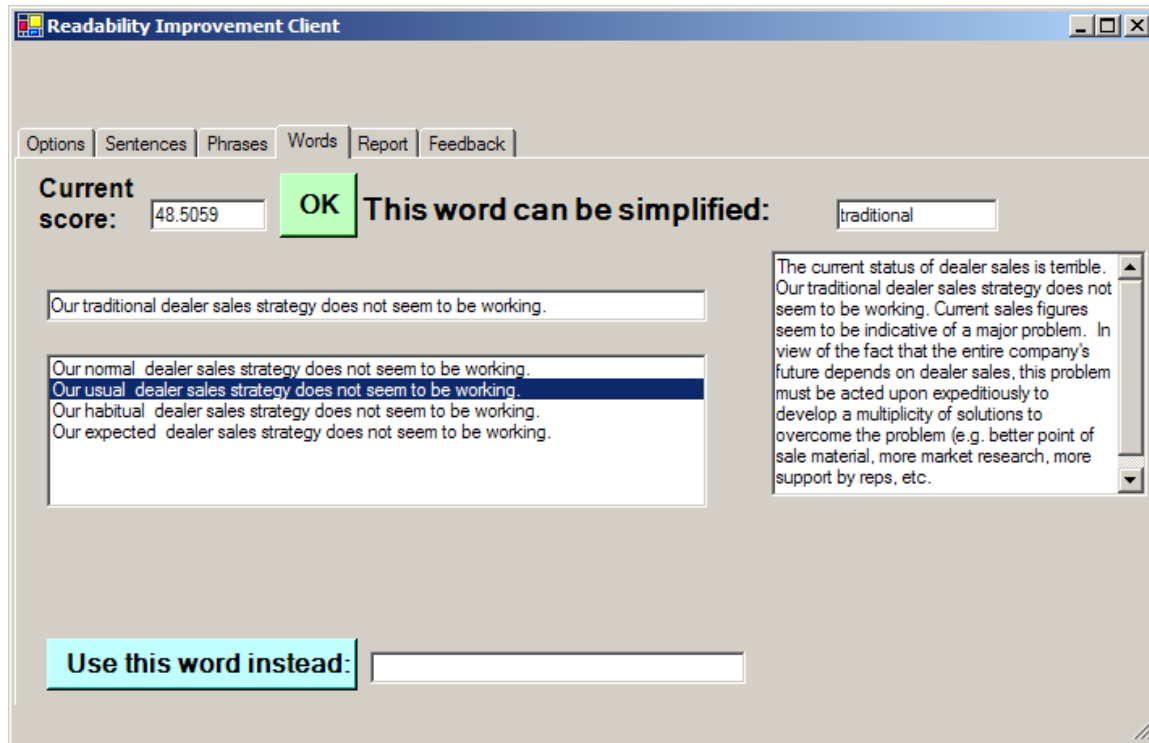


Figure 3.6 GUI for editing and improving words

According to Flesch Reading Ease formula, the reading complexity of a document increases if the text contains words with higher syllable counts. The IDR tool processes the document and identifies all words with higher syllable count. Currently, we are identifying those words whose syllable count is equal to or higher than three. The application provides alternatives for the identified word. The syllable counts of these alternative words are less than the syllable count of original. In the figure 3.6, the application shows that the polysyllabic word “traditional” might be replaced by words “normal”, “usual”, “habitual” or “expected” here. These alternatives are shown inserted into the original sentence so that user can make a decision based on the context.

3.2.3 GUI for restructuring sentences

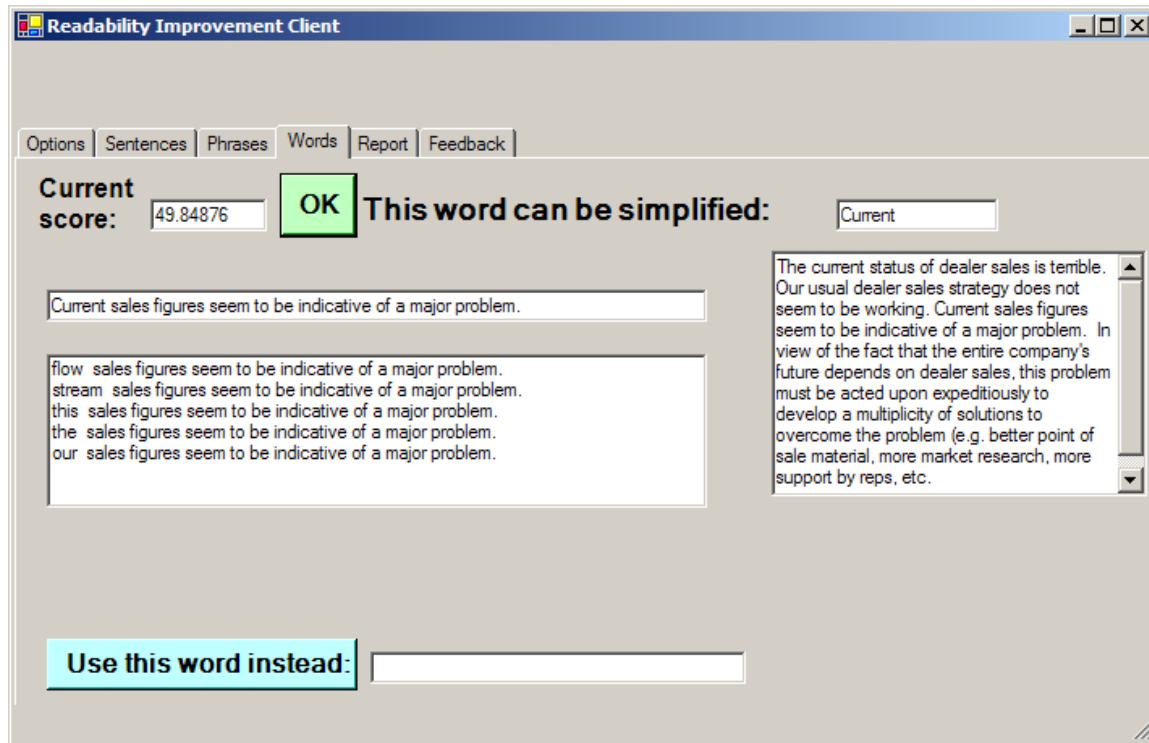


Figure 3.7 GUI for restructuring sentences

In the sentence “Our traditional dealer sales strategy does not seem to be working”, the user replaces the word “traditional” with “usual”. The IDR tool updates the document and it is visible in the text box as shown in Figure 3.7. The readability score is updated as the user edits the document. As shown by Current Score the readability score increases to 49.84 from previous score of 48.50.

3.2.4 GUI for updating sentence

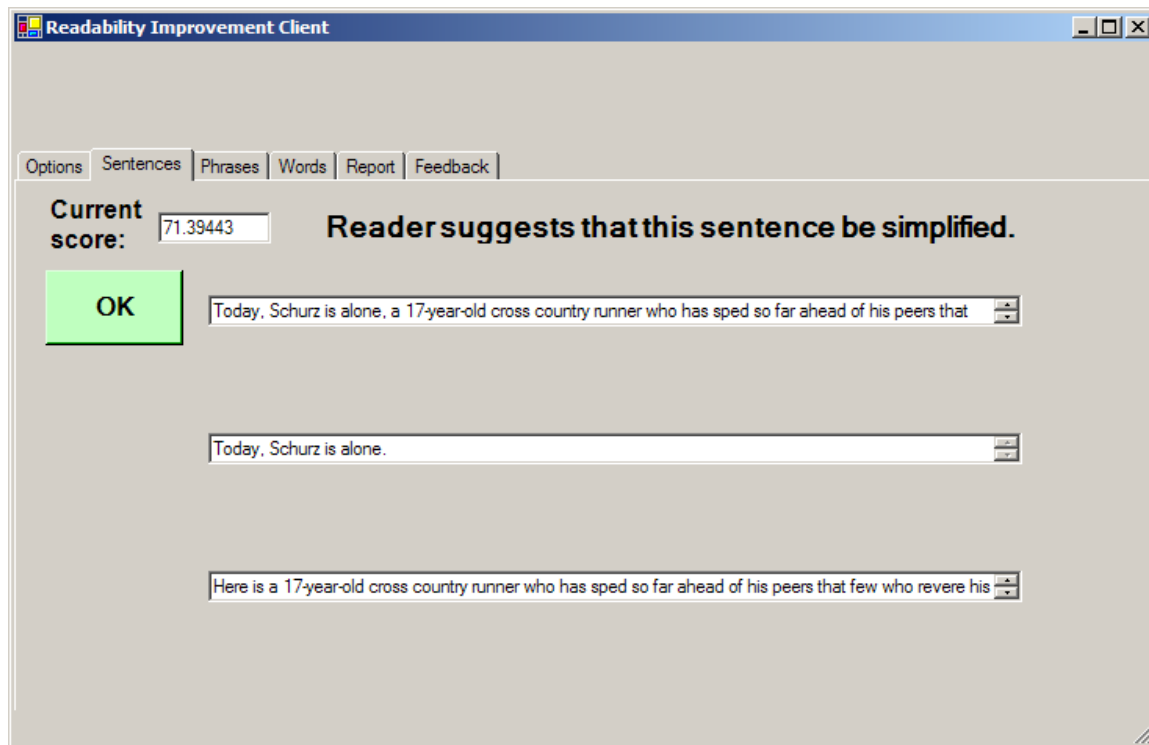


Figure 3.8 GUI for updating sentence

One of the important steps in improving document readability is to reduce the average length of the sentences. IDR tool has the option of displaying the longest sentence and allowing the user to manually divide it into smaller sentences. As shown in the figure, the IDR GUI displays the current score and the longest sentence under the caption "Reader suggests that this sentence be simplified". The longest sentence displayed is

"Today, Schurz is alone, a 17-year-old cross country runner who has sped so far ahead of his peers that few who revere his speed and grace dare guess how far his talents will take him"

The User can edit this long sentence to make it shorter. When a user completes the editing and clicks on "OK", the text is modified with shorter sentences.

3.2.5 GUI displaying modified sentence

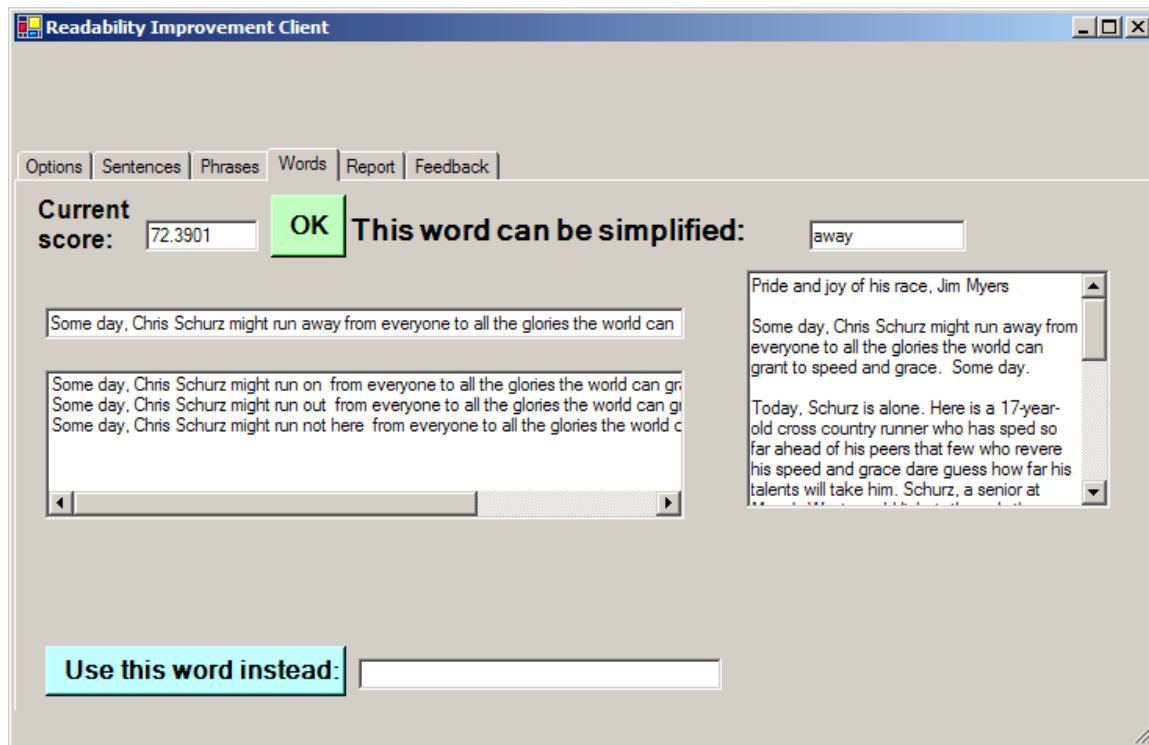


Figure 3.9 GUI displaying modified sentence

The modified document is shown after the user changes the longest sentences.

The sentence

“Today, Schurz is alone, a 17-year-old cross country runner who has sped so far ahead of his peers that few who revere his speed and grace dare guess how far his talents will take him”

has been made shorter with the sentences

“Today, Schurz is alone.”

“He is a 17-year-old cross country runner who has sped so far ahead of his peers that few who revere his speed and grace dare guess how far his talents will take him”

Subsequently the readability scored improved from “71.39443” to “72.3901” as shown in the figure 3.9

3.2.6 GUI for modifying phrases

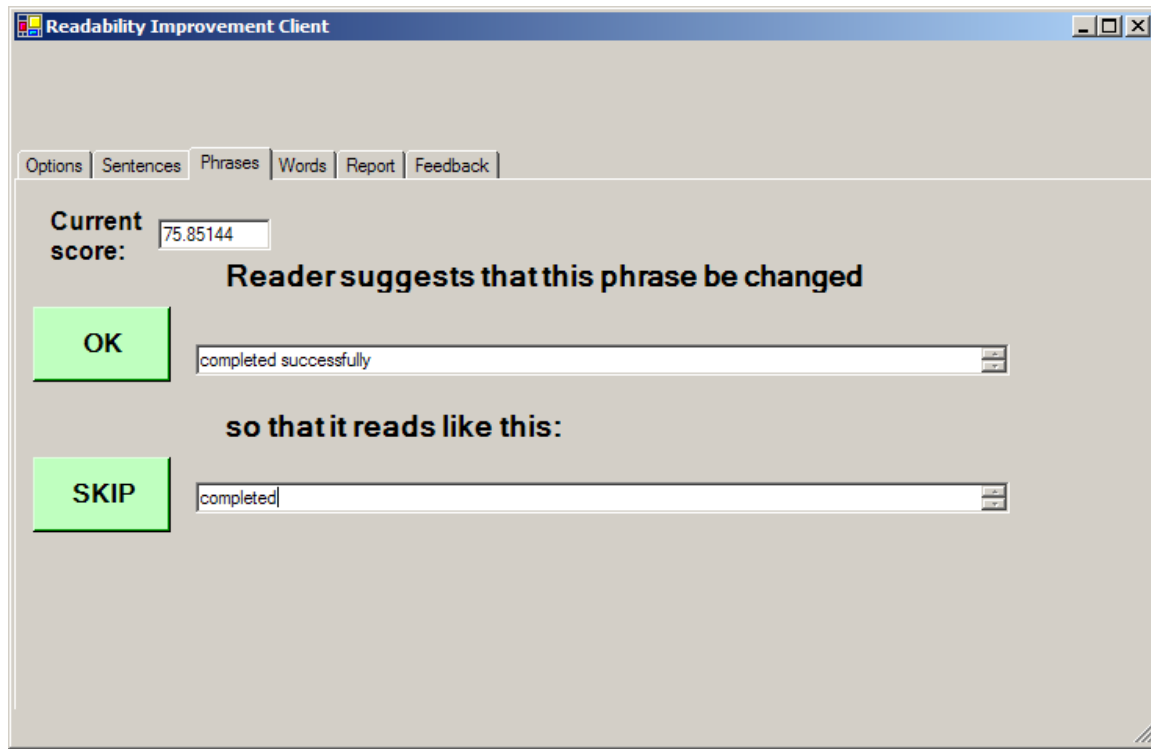


Figure 3.10 GUI for modifying phrases

This option allows user to modify common phrases by deleting words. These words are unlikely to add significant to the text. For example, in the portion of the sentence “completely successfully”, the word “successfully” is not essential. This word can be deleted without affecting the content and thereby making the sentence simpler.

3.2.7 GUI for Report

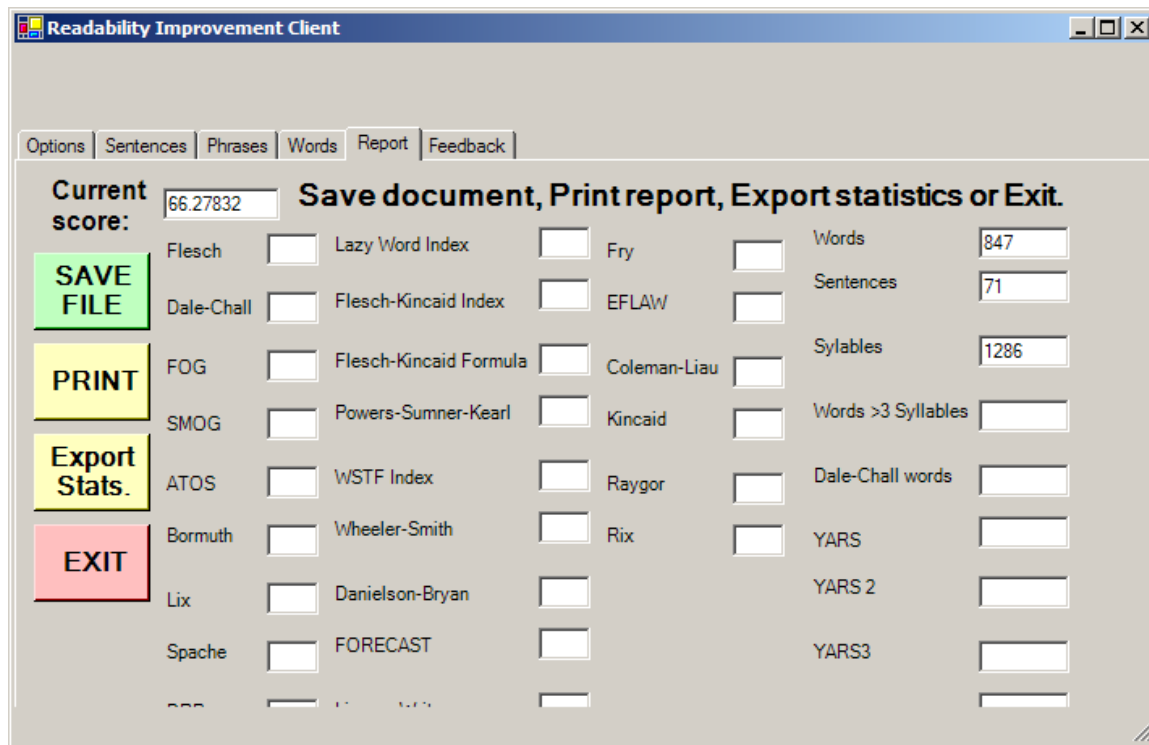


Figure 3.11 GUI for Report

There are a number of options under the section reports. As shown in the figure, a user can save and print the file and also export the statistics to other applications. Currently, we are displaying the statistics related to total number of words and total number of sentences in the document. The report also displays the total syllable counts, which we calculate by adding the syllable count of each of the individual word. We also made provision to report statistics for some of the readability metrics. However, they are not currently implemented.

3.2.8 GUI for user feedback

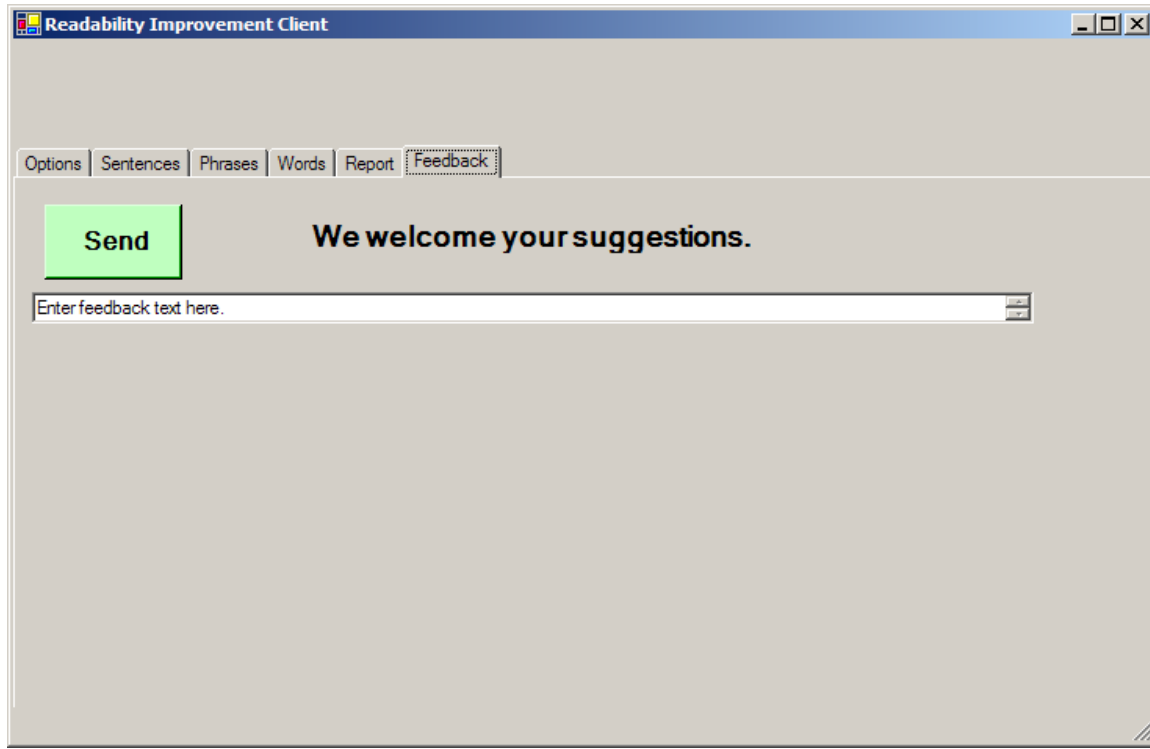


Figure 3.12 GUI for user feedback

The application is also providing the option of user feedback. The user can enter the information and it will be sent to the system administrators. The feedback may be on the structure of the words, for example, a user might not agree with certain synonyms of a word and may suggest other alternatives. This feedback will be considered for updating the lexicon.

3.3 Designing the Client Server application

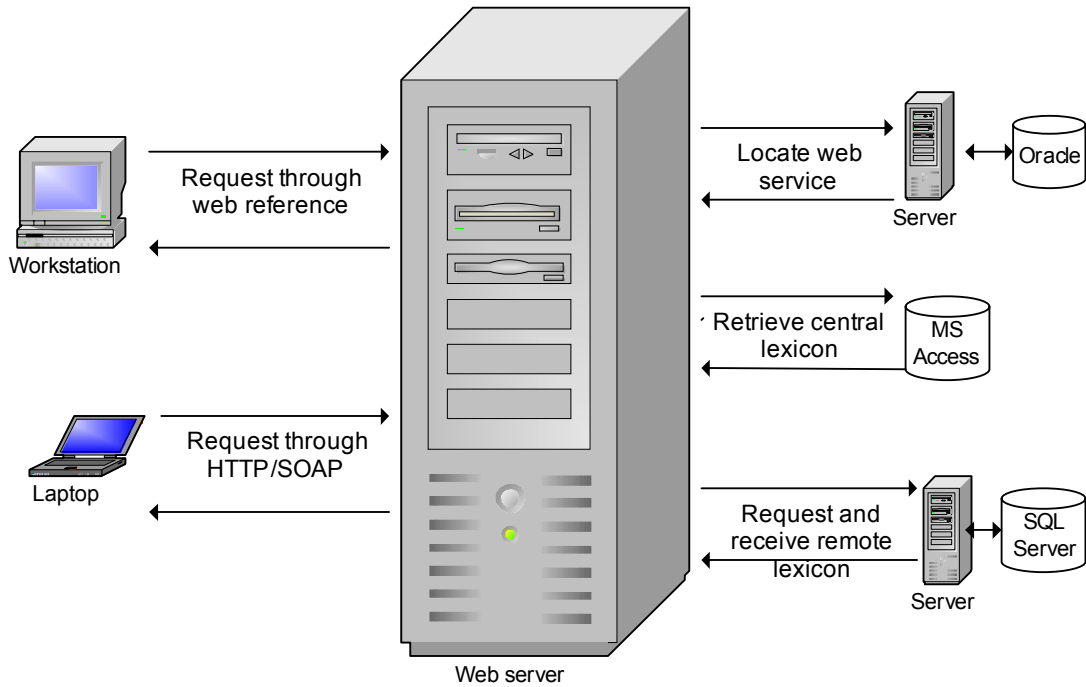


Figure 3.13 Physical flow of data in Client Server Environment

A Client Server system is a solution in which the presentation, application, data manipulation and data layers are distributed between client machines and servers. This type of application is useful for corporate users. The corporate users are a group of users sharing a lexicon. In this architecture, the lexicon is stored in a centralized location. As such users will be able to share the lexicon. As shown in the figure 3.13, the client computer and the Web server are physically located in different machines.

This section describes the implementation of the application with client server architecture. As shown in the figure 3.13, this architecture consists of a set of clients, a web server and a set of web services. The client computer may be a personal computer, workstation or laptop computer. The web server is more powerful than the client

computer. It communicates with the client by returning documents (in a format such as HTML) and data (in formats such as XML). The web server also hosts the Web Services. As we discussed earlier all the components of our application have been hosted as web services. The client application contains references to Web services which are called Web reference. A web reference indicates the location of the web server and also the services exposed by the server. The Web server listens for the client requests, determines the appropriate Web Service for that request, invokes that service and returns the service response to client.

3.3.1 Organization of Web Services

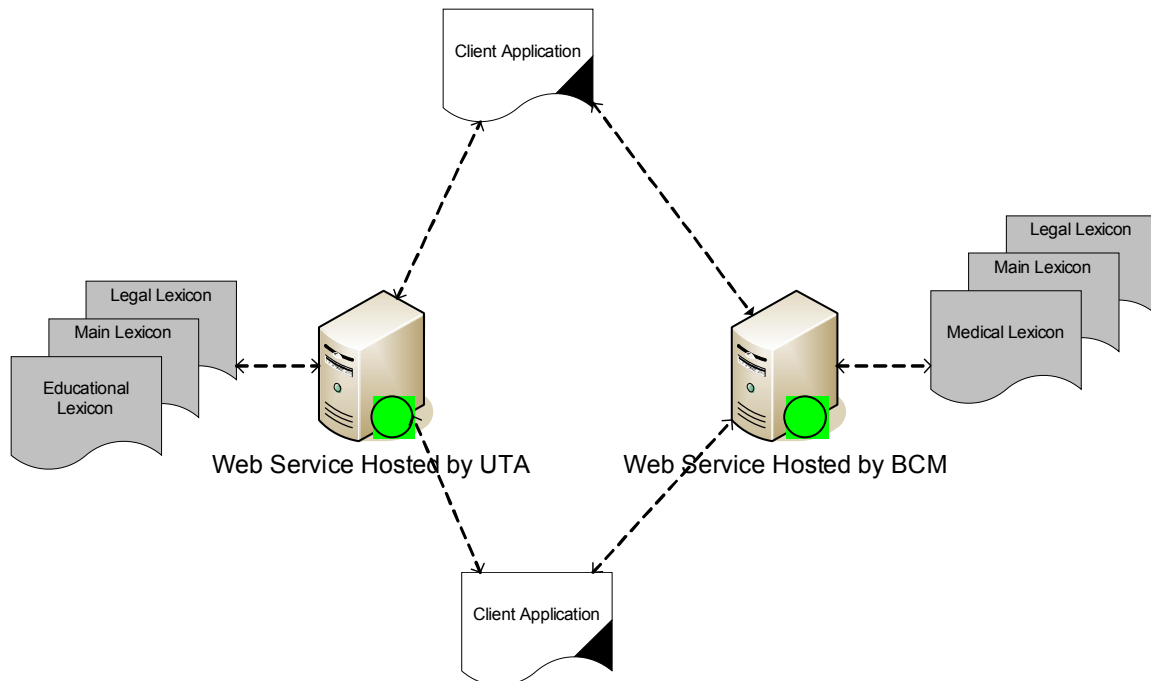


Figure 3.14 Web services hosted by Web Server

Web services are self contained, modular programs that can be published and invoked via the Web. Each Web Service serves a specific function. After a Web service

is deployed, user applications can invoke the methods that are exposed by that Web Service. The biggest benefit of a Web service is its simplicity; it allows pieces of functionality offered by different organizations to work together, exchanging information seamlessly [ASP .NET 06].

As shown in the figure 3.14, users of IDR are benefited by Web Services offered by UTA (University of Texas at Arlington) and BCM (Baylor College of Medicine). For example, BCM might have a rich repository of medical terms that may not be available at UTA. A user from UTA is benefited if the BCM server exposes the medical lexicon. Similarly, a physician at BCM might gain by using an educational lexicon exposed by a UTA Web Service.

For our Client Server architecture, we have decided to use Web Services at the server. We mentioned in the Introduction chapter that there would be different sets of users for our IDR application. We have developed a prototype using the Web Service and at the end of this chapter, we will evaluate the merits of this architecture.

3.3.2 Logical flow of data in Client Server architecture

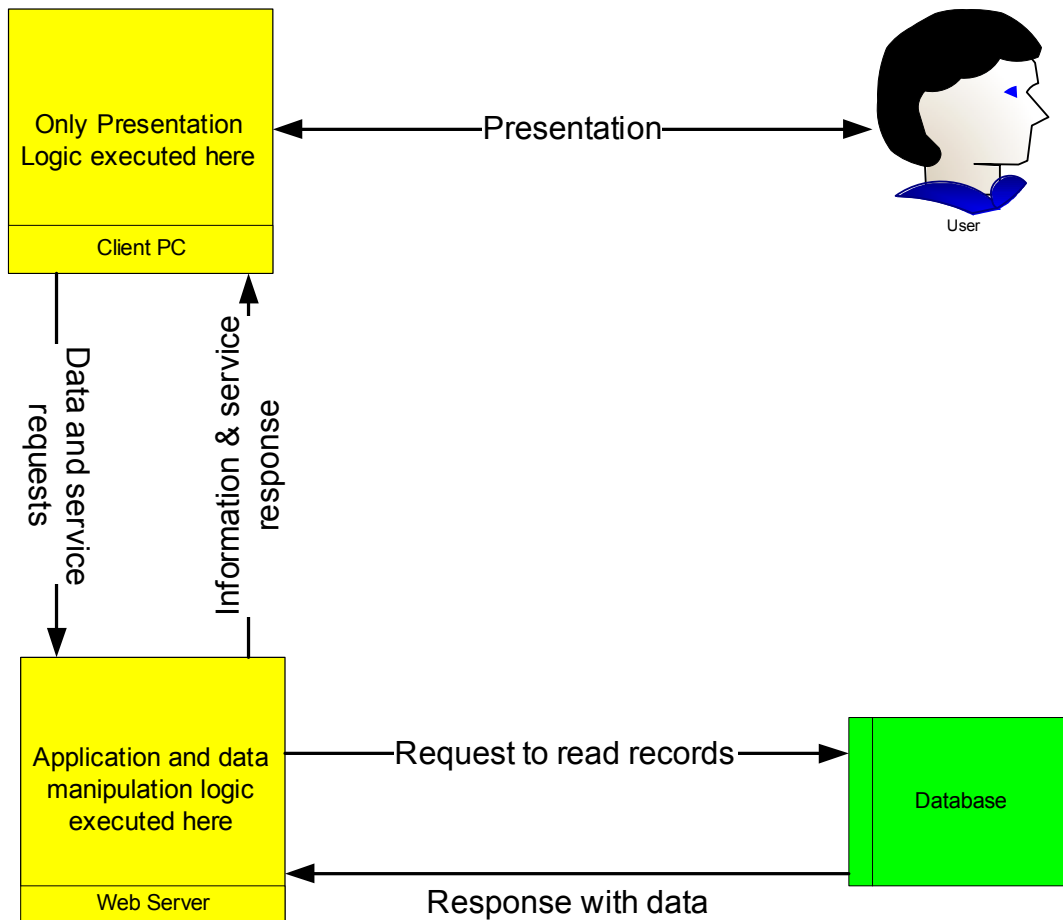


Figure 3.15 Logical flow in Client server

As shown in the figure, the application and data manipulation logic are placed on the web server and the presentation logic is placed on the client. We define the following terms;

- Data manipulation logic – This logic is related to connecting, retrieving or updating the database.

- Application logic – This logic is specific to the application. For example, we have defined rules for parsing. These rules are part of the application logic.
- Presentation logic – This is related to the presentation of the graphical user interface.

In the Client Server system, the client executes a minimum of overall system's components. Only the user interface and some relatively stable or personal application logic needs to be executed on the client. The Server performs the majority of the functionality.

3.3.3 Client in Client Server application

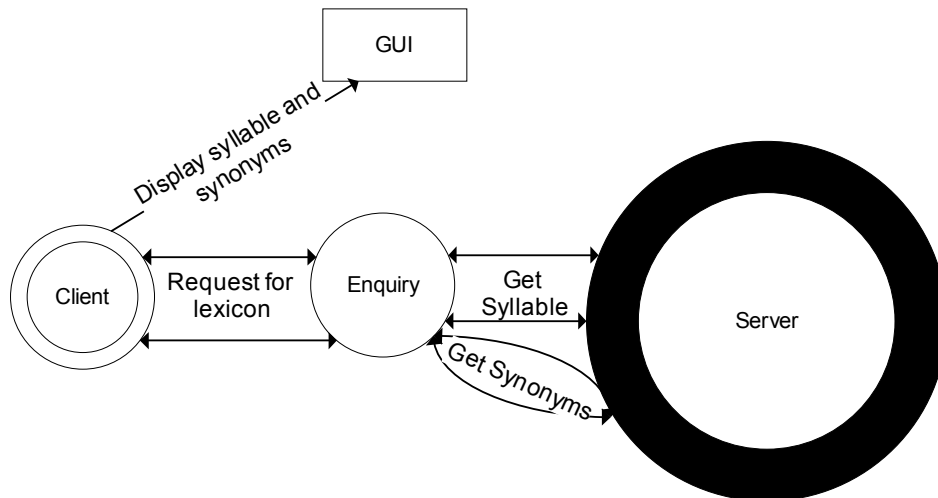


Figure 3.16 Client in Client Server

The Client is a lightweight process that provides a graphical user interface to the user. The Client initiates actions based on the commands issued by the user. For

example, when the user issues a command for the longest sentence, Client application invokes the necessary method for computation and displays it in the GUI.

One of the major tasks of the application is loading the lexicon. The application is designed in such a way that the lexicon is available for query as the user continues to use the GUI. The lexicon is loaded in the user's memory space through a data table. This has ensured that data exchange between client and server is not needed subsequently. If user needs the list of synonyms for a particular word, the application can retrieve the list from the data table and the response is found to be faster. This works well as long as the lexicon is of manageable size and it resides in the client machine. However as the lexicon size increases and if it is physically located in a different machine, this will cause considerable time to load in user's machine. Moreover if the user is going to use the central or a corporate lexicon, this may cause significant delay. We have decided to delegate this work to another process which we call the enquiry process.

We are also using the term 'vertical slice' of the database. A vertical slice is the column content of entire rows of a table. We observed that during document parsing, the Parser needs only certain columns of the lexicon. They are the syllable count, flag byte and the words. We grouped them into a vertical slice called syllable. We have defined another vertical slice called synonyms that consists of synonym list of the lexicon. The purpose of the vertical slice is to retrieve column of lexicon as needed. The synonym lists are not needed during the parsing of the text. We deferred retrieving this

vertical slice at this stage. We load them when the word is being considered for replacement.

The Enquiry process: The enquiry process manages the lexicon in response to client query. The purpose of the enquiry process is to manage the retrieval of lexicon. As shown in the figure 3.16, the enquiry retrieves the lexicon in vertical slice of syllable and vertical slice of synonyms. We elaborate the task of enquiry in response to user action as follows.

The User starts the application: When the user starts the application, the client application is loaded. At the same time, the Client spawns the Enquiry process. The Enquiry process connects to the Web Server and requests the lexicon. In response to the request, The Web Server returns the lexicon. At this instance, the Enquiry process retrieves only the syllable vertical slice and makes this available to client.

The User Action to Process the document: The parsing process is called in response to this command. The Parser needs the syllable counts of the words and these are already available as syllable vertical slice. Based on this data the readability formula is processed and displayed.

User Editing: The user can edit the words with synonyms as described earlier. As the user begins editing, the client application invokes the Enquiry process to retrieve the synonyms of the edited word. The Enquiry process connects and retrieves the synonyms vertical slice and makes it available as the user continues editing the document.

It is expected that the slicing of the data will improve the performance of the Client Server application in terms of response time. Once loaded the vertical slice will remain in memory till the application is closed. The Parser can reuse this for processing of other text files. This approach will also benefit if we expand the database. For example, we may include the frequency count of words as columns of lexicon. We will have a vertical slice of frequency count and we will retrieve this information when it is required by application.

3.3.4 Server in Client Server application

In a Windows environment, a web service is hosted by the Internet Information Server (IIS). IIS, in turn, is a component of the Web Server. IIS contains one or more virtual directories each hosting a particular Web Service. A web service is responsible for interacting with a particular lexicon. As such, we can have a number of web services.

Web services are designed for interoperable applications. We are using the term interoperable to mean that a client can invoke a web service regardless of client's hardware or software. For example, an interoperable web service running on WebLogic Server on a Sun Microsystems computer can be invoked from a Microsoft .NET Web Service client written in Visual Basic. It is possible that these applications use different data types to communicate with the service. A proxy class is used to facilitate these communications. The proxy class sits in between the consumer application (marked as client in the figure 3.18) and the Web service. It offers the same methods as the Web

Service, automatically translating any call to one of those methods into the XML request and then translating the XML response back into a simple return value.

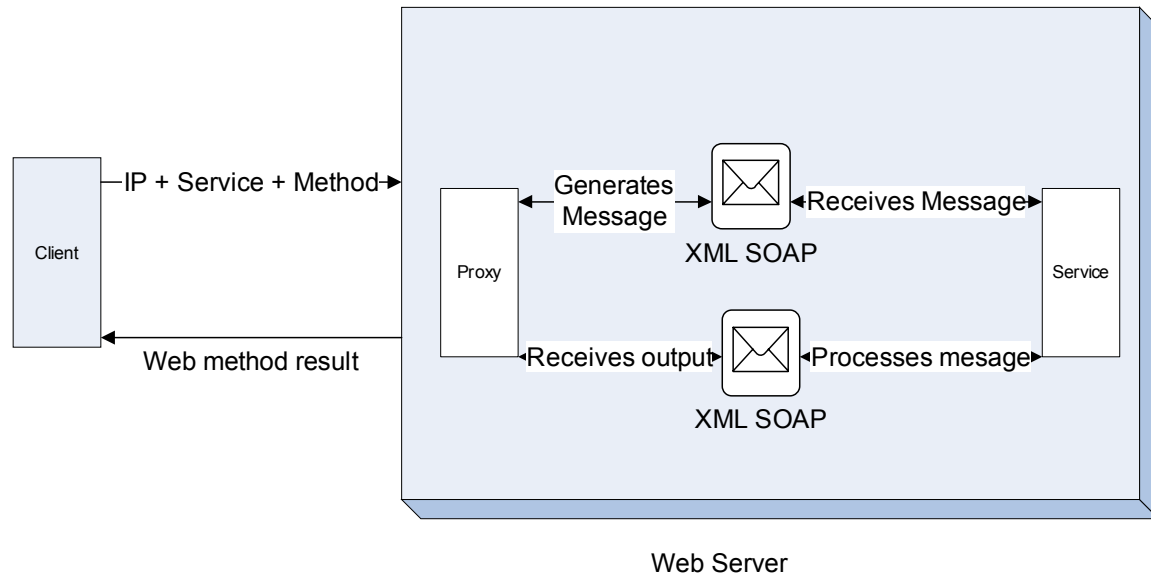


Figure 3.17 Server in Client Server

The following sequence of events takes place when an application calls a Web Service:

- A client call to the web service consists of two parts. The first part contains the IP address and the second part contains the Service name. When a client calls a web service, a corresponding proxy object is instantiated
- The proxy object converts the method request and input parameters into an XML message and relays them to the web service.
- The web service receives and processes the input message and generates output.
- The Proxy receives the output and returns the result in a format expected by the client.

The introduction of the proxy has ensured that the client application is not affected by the underlying implementation of Service. A Service may be developed in Java, however the proxy objects ensured that this can be used by a Microsoft Windows based application.

Table 3.1 Comparing Web Services with other standards

Characteristics	CORBA	DCOM	Web Services
Method calling	InterORB Protocol	Remote Procedure call	HTTP
Firewall Friendly	No	No	Yes
Cross Platform	Partly	No	Yes
Implementation cost	High	High	Low

The web service has a number of advantages over other distributed component model. The other models like DCOM (Distributed Component Object Model), Common Object Request Broker Architecture (CORBA) and Java Remote Method Invocation (RMI) are suitable for intranets where the platform is homogenous and there is no firewall intervention. For our implementation these models are not suitable because the remote lexicon can reside on any platform and they should be accessible from anywhere.

Web service uses a Protocol called Simple Object Access (SOAP) to transmit XML data over a network. XML messages are wrapped within a SOAP envelope and transmitted via HTTP in the form of SOAP requests and SOAP responses. SOAP offers a number of advantages:

- Firewall permeability: SOAP transmits data through HTTP protocol. The Firewall normally allows this protocol to pass through. This has ensured that IDR application can communicate with remote and local clients transparently.
- Enforcing security: The structure of SOAP has greatly helped to ensure security to our web service. SOAP data are XML based. For our Internet based application we have enforced the policy that each SOAP packet will contain user credential information along with the data. We have developed a XML parser to validate the XML data. Whenever a request is made, this request is validated at the receiving end. This has ensured that only request with valid user credentials are allowed.
- Error handling: SOAP has inherent mechanisms whereby it can handle errors and pass the error to the caller. If the server cannot handle a service request, SOAP returns a fault message. The fault message is the root element of the Body of the response. It contains the human readable description of the error. This has enabled the client to handle errors gracefully and thus error handling became transparent.
- Low cost of implementation: Web services are found to be cost effective in terms of development life cycle and maintenance. Microsoft has provided tools for rapid application development environment and we have been able to take advantage of that and build our web services efficiently in a short time span.

3.4 Designing the distributed architecture using Software Agent

Software Agents are intelligent entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in doing so, employ some knowledge or representation of the user's goal or desires. The primary purpose of a Software Agent is to identify a specific need that the user has and to act on that need. An Agent can automate many user tasks, such as performing book keeping on local or remote data. In our application, we will use a software agent to assist the remote user in maintaining a local repository of the lexicon available from central or corporate servers.

One of the major goals of our readability application is to provide the user precise, up-to-date data. This involves maintaining a local repository of the lexicon available from the corporate or central servers. Our aim is to provide a remote user's laptop with data from anywhere so that the user can benefit from a large set of lexicons.

The corporate users share a common lexicon. For this type of users it is essential that we provide mechanisms to exchange lexicon data. We will use the Software Agent to assist users in this task. It is expected that a Software Agent will perform the following:

- Execute independently from the main application
- Run and perform periodic updates on local, central and corporate lexicon

In our application the Software Agent will perform as a Window Service.

3.4.1 Developing the Software Agent

One of the major goals of our IDR application is to keep the corporate users with precise, up to date data. The IDR tool in the distributed environment has not provided functionality to allow the user to enter a new entry in the lexicon. We developed a new technique to achieve this objective:

- We will maintain two sets of lexicons, local and corporate
- A local lexicon is installed in the user machine at the time of application set up.
- A corporate lexicon is installed at the Server. Initially it is empty and is updated dynamically.
- A Software Agent called Remote Agent is also set up in the user's machine. The Remote Agent monitors changes in the local lexicon. Whenever the user enters a new entry into the local lexicon an event is triggered. The Remote Agent notifies the server about the update. This information is transmitted through XML.
- A Software Agent called Server Agent also runs in the server. The job of the Server Agent is to monitor notifications from Remote Agents; update the corporate lexicon in case of modification and also to distribute this updated information to the other remote agents.

3.4.2 Architecture of the Software Agent

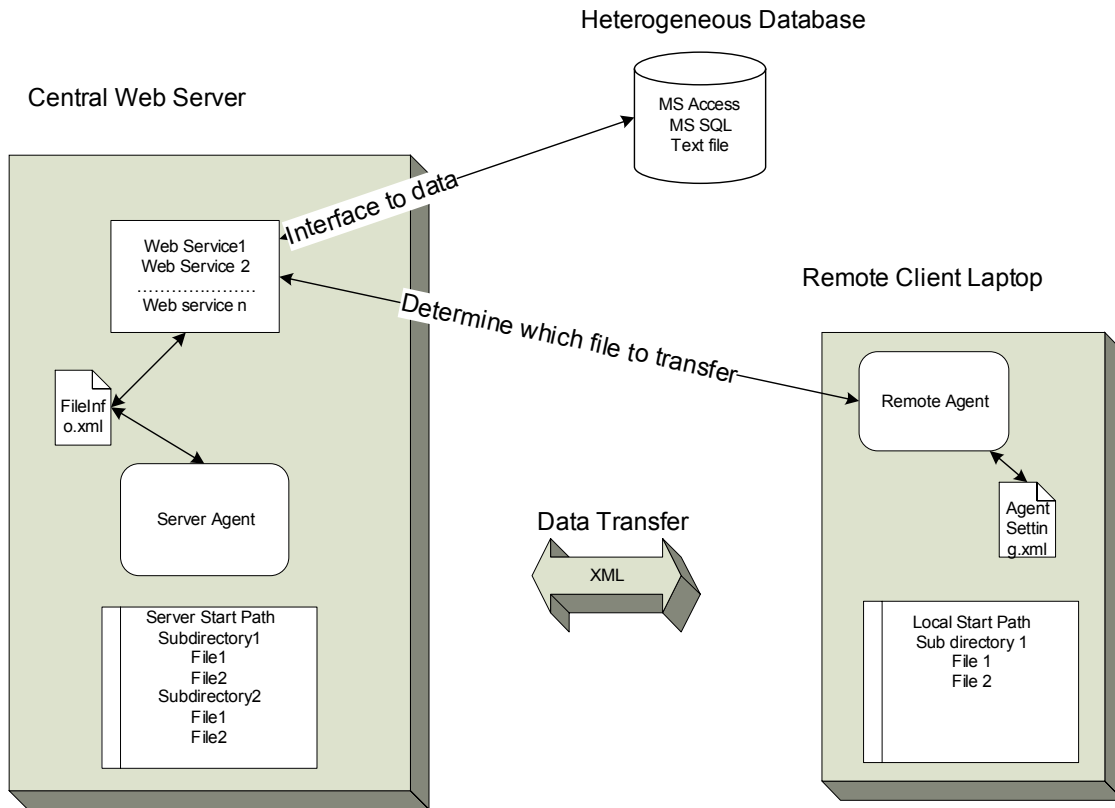


Figure 3.18 Data flow in Software Agent solution

As shown in figure 3.18, we have a single Server Agent running in our centralized web server and Remote agents each running in individual user machines.

Both the Server Agent and Remote Agent keep a configuration file maintained in XML format. We are using the configuration file to keep track of the update or modification requested by the user. A user may find that the synonyms of a certain word are not available in the current lexicon. He enters the synonyms through our IDR tool and our application in turn will enter these in the configuration file. The Remote Agent running

in the client machine detects the new entry and it will notify the server to incorporate this into the corporate lexicon located at the server.

The following is a segment of configuration file located in client machine.

```
<xml version="1.0" encoding="utf-8"?>
<agentserverpath last updated ="7/23/2004 12:41:23 PM">
<lexicon="Regular">
    <createdate>3/20/2006 4:20 PM</createdate>
    <modifieddate> </modifieddate>
    <Modifiedflag>N</Updateflag>
    <Word "jubilation" Syllable Count ="4" >
        <Synonyms> elation </Synonyms>
        <Synonyms> triumph </Synonyms>
    </Word>
</subdirectory1a>
</agentserverpath>
```

The entries in the configuration file indicate that the client has entered the word “jubilation” with the corresponding synonyms. These entries are intended for the corporate lexicon “Regular” located in the server. The value of “Modified flag” is “N”. This indicates that this word and the Synonyms are not yet transmitted to the server. When the Remote agent detects this new entry, it will start transmitting this information to server. On completion Remote Agent sets the “Modified flag” to “Y”. It also enters the “modified date”. The configuration file is updated as follows

```
<xml version="1.0" encoding="utf-8"?>
<agentserverpath last updated ="7/23/2004 12:41:23 PM">
<lexicon="Regular">
    <createdate>3/20/2006 4:20 PM</createdate>
    <modifieddate> 3/25/2006 6:20 PM </modifieddate>
    <Modifiedflag>Y</Updateflag>
    <Word "jubilation" Syllable Count ="4" >
        <Synonyms> elation </Synonyms>
        <Synonyms> triumph </Synonyms>
    </Word>
</subdirectory1a>
```

The “Modified flag” ensures that this information is not transmitted twice. This will also ensure that configuration file does not grow bigger. The Remote Agent can delete entries whose “Modified flag” is “Y”.

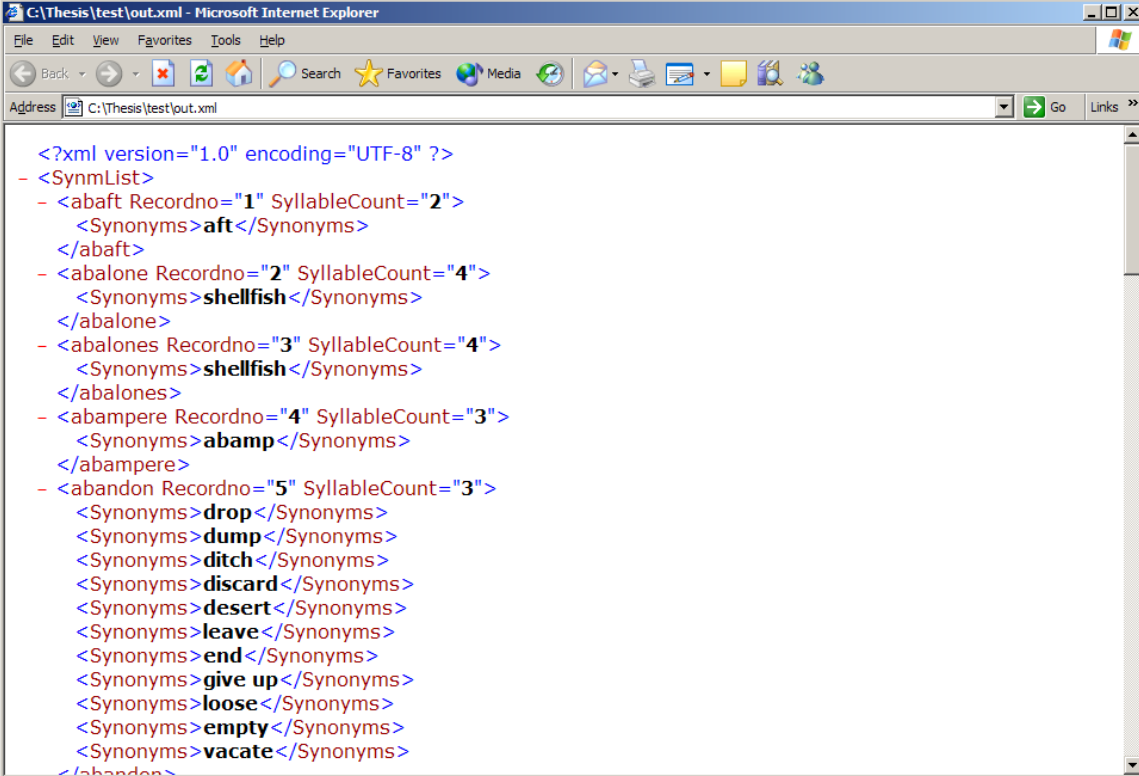
The following sequence of events takes place when a user in a client machine enters a new entry in the local lexicon:

- The Remote Agent detects the change and updates its configuration file.
- The client machine connects to the server.
- The Server Agent running in the server detects the updated configuration file of the Remote Agent. Based on the user’s action it will determine whether updated information needs to be incorporated in the corporate lexicon.
- If the Server Agent decides to incorporate the lexicon data of the client, it will update its own configuration file. This configuration file is updated to let other clients know about the changes.
- When the Client machine connects to the server, the Remote Agent kicks off its processing. It receives a copy of the Server Agent’s configuration file. It will examine the last updated date and determine whether a change has taken place since the last time it checked. In case of update, the Remote Agent will download and add the new information.

3.4.3 XML for Software Agent

A lexicon can be a Microsoft Access table, an Oracle database table and even a flat file. Because of this heterogeneous nature of data, we have to agree on a common protocol to communicate between different data sources. We have decided to use XML. XML is a universal format that can be shared across diverse data sources. XML is found to be convenient to transmit and update data both in the client and the server.

To accomplish our objective for the Software Agent technique, we have decided to map the lexicon data to XML format. The figure 3.19 shows a segment of data from Microsoft Access table in XML format.



```
<?xml version="1.0" encoding="UTF-8" ?>
- <SynmList>
- <abaft Recordno="1" SyllableCount="2">
  <Synonyms>aft</Synonyms>
</abaft>
- <abalone Recordno="2" SyllableCount="4">
  <Synonyms>shellfish</Synonyms>
</abalone>
- <abalones Recordno="3" SyllableCount="4">
  <Synonyms>shellfish</Synonyms>
</abalones>
- <abampere Recordno="4" SyllableCount="3">
  <Synonyms>abamp</Synonyms>
</abampere>
- <abandon Recordno="5" SyllableCount="3">
  <Synonyms>drop</Synonyms>
  <Synonyms>dump</Synonyms>
  <Synonyms>ditch</Synonyms>
  <Synonyms>discard</Synonyms>
  <Synonyms>desert</Synonyms>
  <Synonyms>leave</Synonyms>
  <Synonyms>end</Synonyms>
  <Synonyms>give up</Synonyms>
  <Synonyms>loose</Synonyms>
  <Synonyms>empty</Synonyms>
  <Synonyms>vacate</Synonyms>
</abandon>
```

Figure 3.19 Lexicon data in XML format

As shown in the figure 3.19, a lexicon entry can conveniently be represented in XML format. The data represented in XML can be retrieved using the Document Object Model. This has ensured efficient update, add or delete operations in lexicon data. All that is required of the receiving system is the ability to parse the XML data and act on it. The performance of the system may be slightly affected because of the parsing; however we feel that gains of universally acceptable data outweighs this impact.

3.4.4 Logical flow of data in Software Agent

In the Software Agent implementation the lexicon is represented entirely on XML. As such, the application has to access this repository of XML. We will discuss the trade off using XML data representation when we compare the performance of our IDR tool in Windows stand-alone, Client Server, Software Agent and Web environments. The following diagram shows the logical flow of information when we are representing data in XML format:

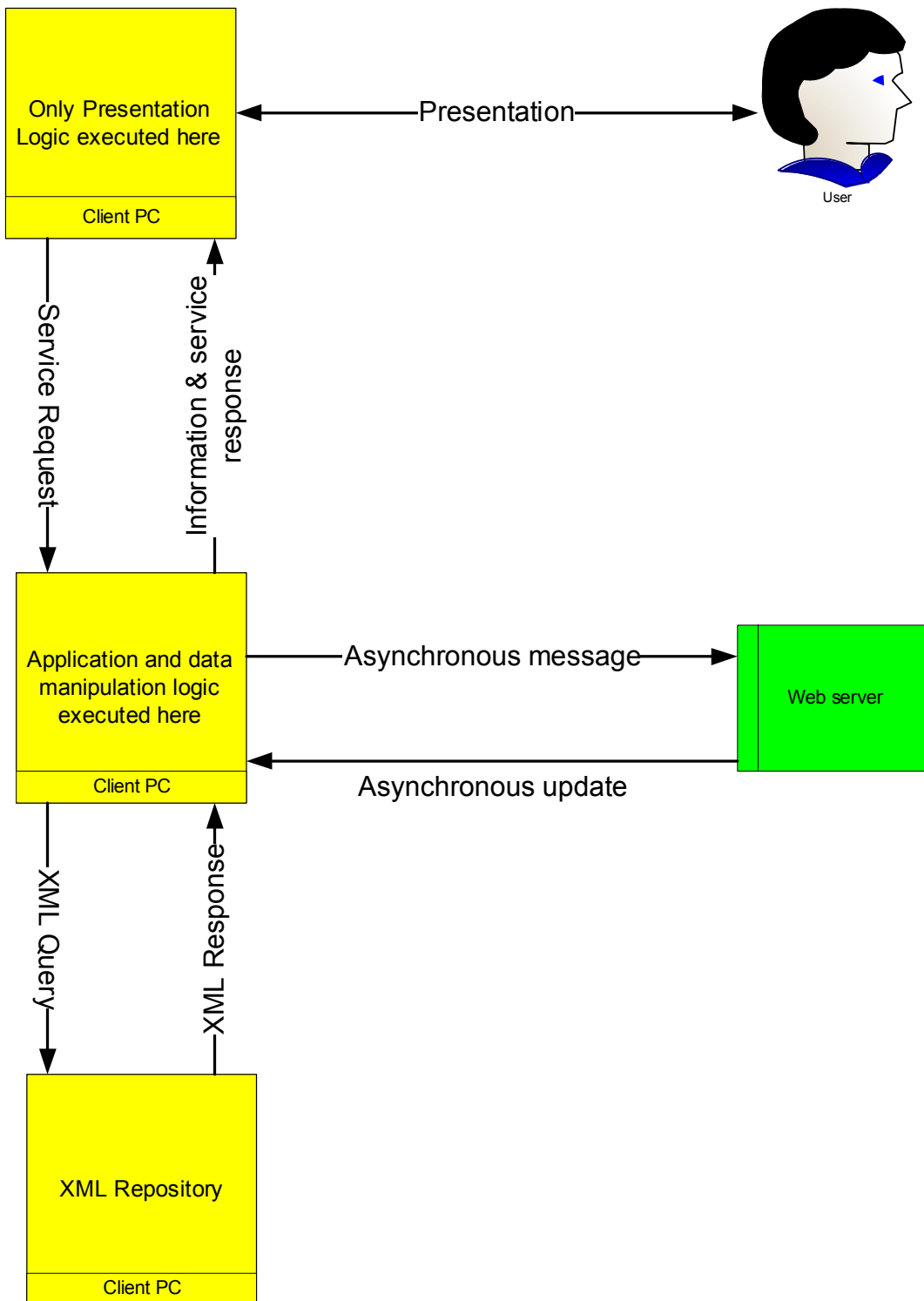
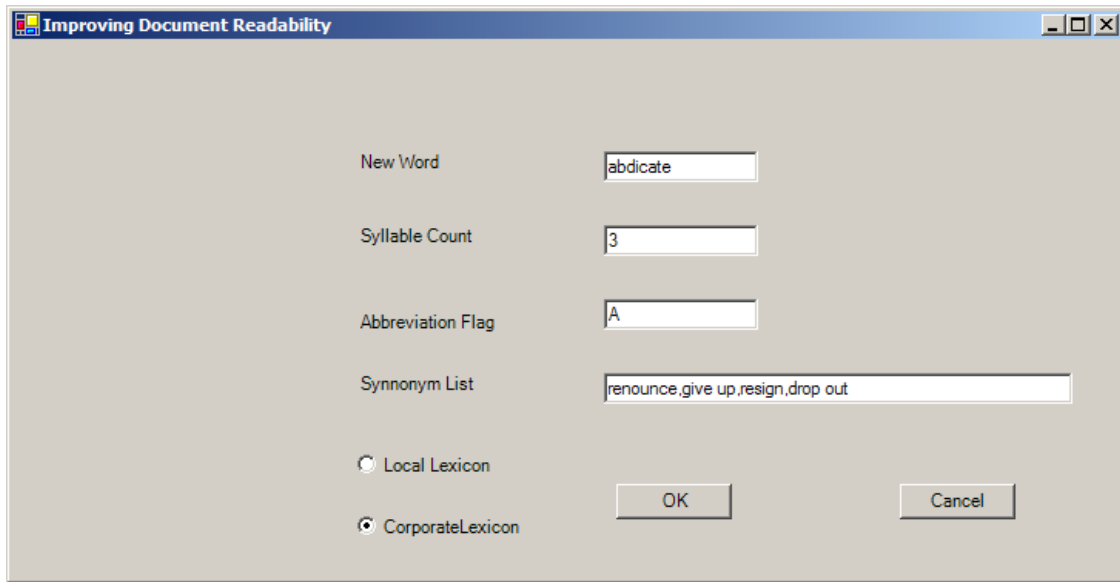


Figure 3.20 Logical flow of data in Software Agent

3.4.5 Updating lexicon with Remote Agent



New Word	<input type="text" value="abdicate"/>
Syllable Count	<input type="text" value="3"/>
Abbreviation Flag	<input type="text" value="A"/>
Synonym List	<input type="text" value="renounce,give up,resign,drop out"/>
<input type="radio"/> Local Lexicon	
<input checked="" type="radio"/> CorporateLexicon	
	<input type="button" value="OK"/>
	<input type="button" value="Cancel"/>

Figure 3.21 Updating lexicon with Remote Agent

Figure 3.21 is a screen shot showing user entering information intended for a corporate lexicon. The user found that the word “abdicate” is not in the lexicon. He is adding that and related data. The remote agent running in his machine finds this update and notifies the Server. In this way, the Software Agent technique can significantly improve the content of lexicon.

3.5 Designing the Internet based architecture

Developing a web interface for a client server based application is an attractive proposition. A user can run the entire application in a Web browser. The fact that anyone from anywhere can access the application is the most beneficial aspect of this type of system.

However, there are a number of major issues that we had to consider before developing the web interface for our application. In Web application, most of the time the application logic is executed on the server. The code for this logic is executed on the web server. The server responds by returning the result to client. A frequent request and response will affect the performance of the application.

Another aspect is that the application has to ensure that user information is maintained in the Server. Normally, whenever an HTML page is rendered, the web object is destroyed and all client specific information is discarded.

3.5.1 Logical flow of Data in a Web Application

An Internet based computing system is a multiered solution in which the presentation and presentation logic layers are implemented in Client side web browsers using content downloaded from a web server. The presentation logic layer then connects to the web server for application logic and data. We show this flow of data in the figure 3.22.

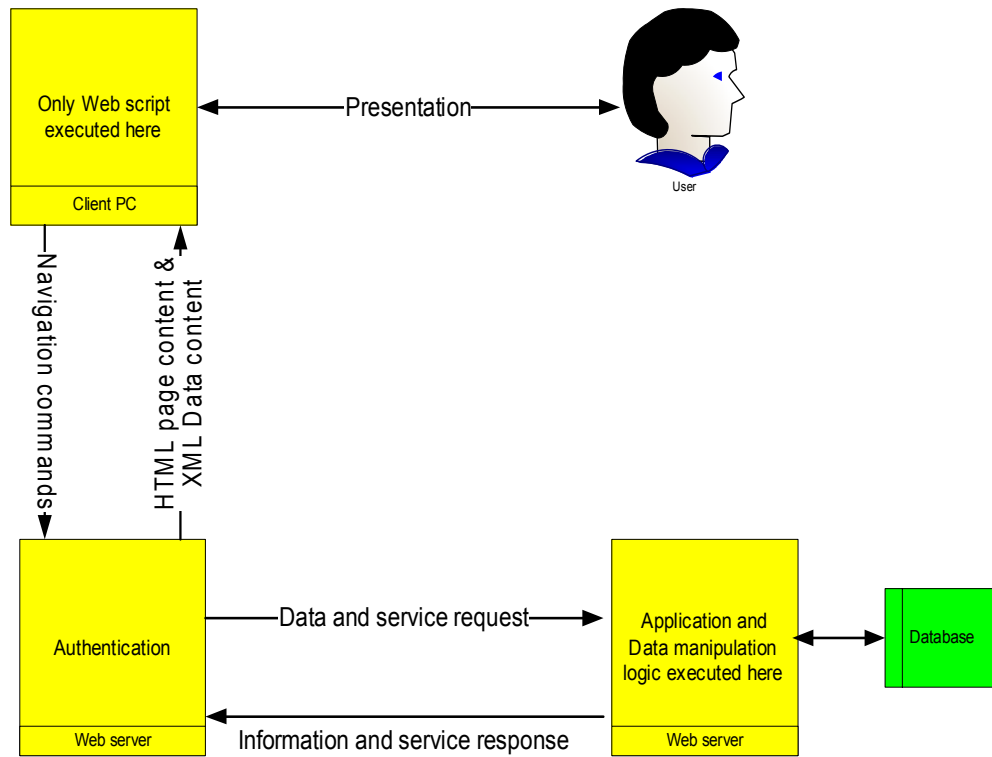


Figure 3.22 Logical flow of Data in a Web Application

There are a number of languages or Scripts that can be used to develop the presentation and presentation logic layers in Internet based computing. HTML or Hypertext Markup Language is used to create the pages that run in the browser. XML is another standard that allows developers to define the structure of the data to be passed to the Web pages. Scripting languages like VBScript or JavaScript are useful for implementing presentation or the application logic in the client.

In the previous chapter, we mentioned the stateless nature of an Internet application. This has implied that we have to transfer all our state data over what is typically the slowest link in our system: the link between the user's browser and the Web Server. Moreover, we are transferring that state twice for each page, from the

server to the browser and from the browser back to the server. Obviously, this can have serious performance implications over a slow network link (like a modem). Considering this, we have redesigned part our application as shown below.

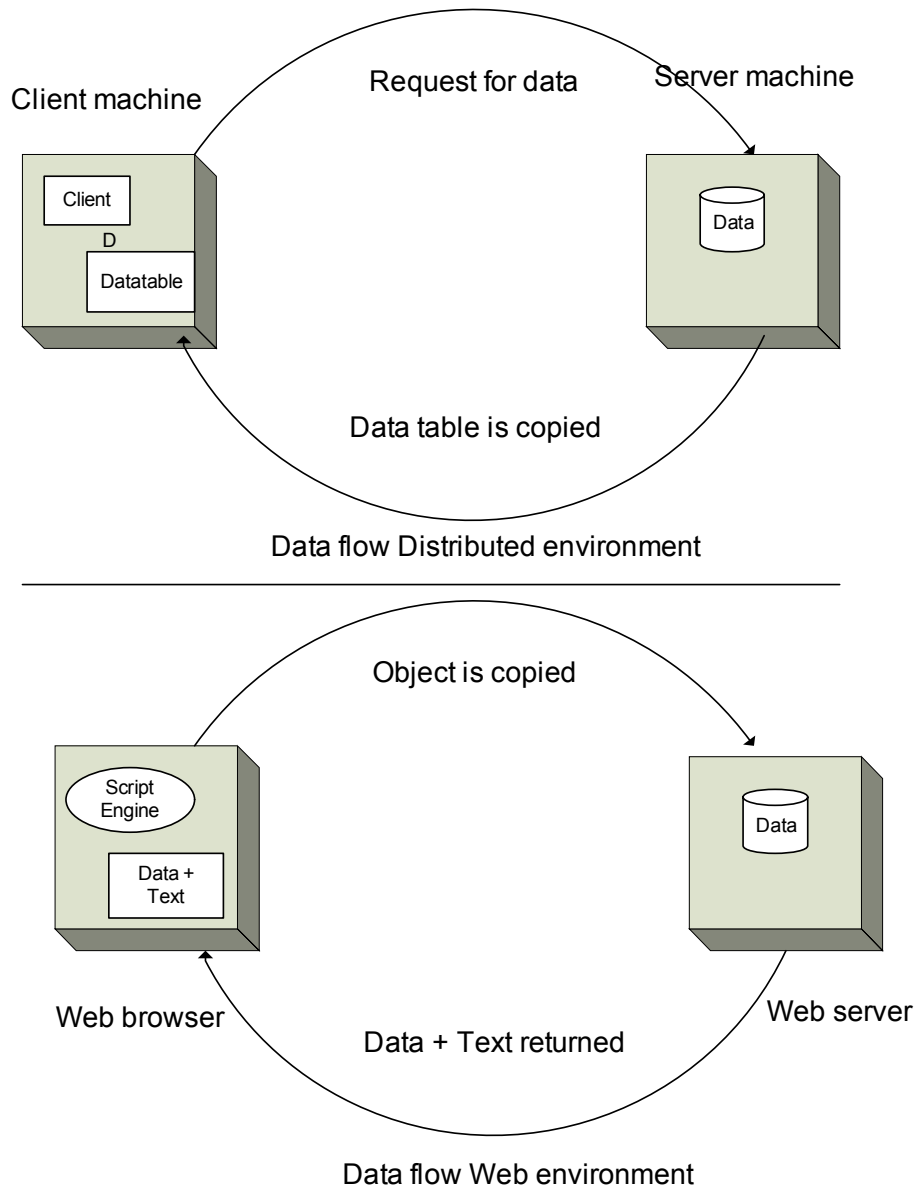


Figure 3.23 Data flow in Web vs. Distributed environment

As shown in the figure 3.23, we made a substantial change in the architecture of this version of the application. In the previous design of client server or the Windows stand-alone application, a connection is established with the database and the entire lexicon is transferred.

While developing the web interface, this approach was avoided. Instead of accessing the entire lexicon, the Web client will retrieve only that information from the database that is required for processing the text. This is accomplished by sending the text content of the input file to the Server. The following sequences of operations take place when a user selects a text file for processing in our Web version of the IDR application:

- The Web interface displays a dialog box; the user can select an input file from his local or network machine. Once the file is selected, the user can click on the upload button to upload the file to be processed in the server.
- When the user selects the process button, the server starts processing the file. The parser will process the text as explained before. The parser object running in the server processes this text and separates the content into words. It also creates a text file containing these words and invokes a database-stored procedure with this text file as a parameter.
- This specific stored procedure accepts this file as input. It iterates over all the words and for each of them retrieves the syllable count and synonym list from the lexicon table. Once the iteration is over all the

input words contain the required information. The Web server sends this information back to the client.

- The client Script can use this information to calculate the readability score.

In this way it is ensured that only the minimum volume of data traffic is transported between client and the server. This design has significantly improved the performance of the application.

This architecture has also ensured that traffic between the Web Browser and Server is light. Once the Web server sends the information back to the client, it is the Script Engine that is responsible for further processing. Here we are describing the Script Engine running in the client machine. The Script is a program written in a Programming language like VBScript or JavaScript. Scripts are usually embedded in the HTML or ASP pages. The ASP pages run in the Web Server while HTML Script runs in the client. The engine component associated with a Web Browser interprets and run the script.

For our IDR Web application, we have assigned some of the processing to the script. Some of the tasks like calculation of readability score and sentence restructuring are repetitive in nature. In our Web Interface, client side scripting performs these tasks.

3.5.2 Client Script processing

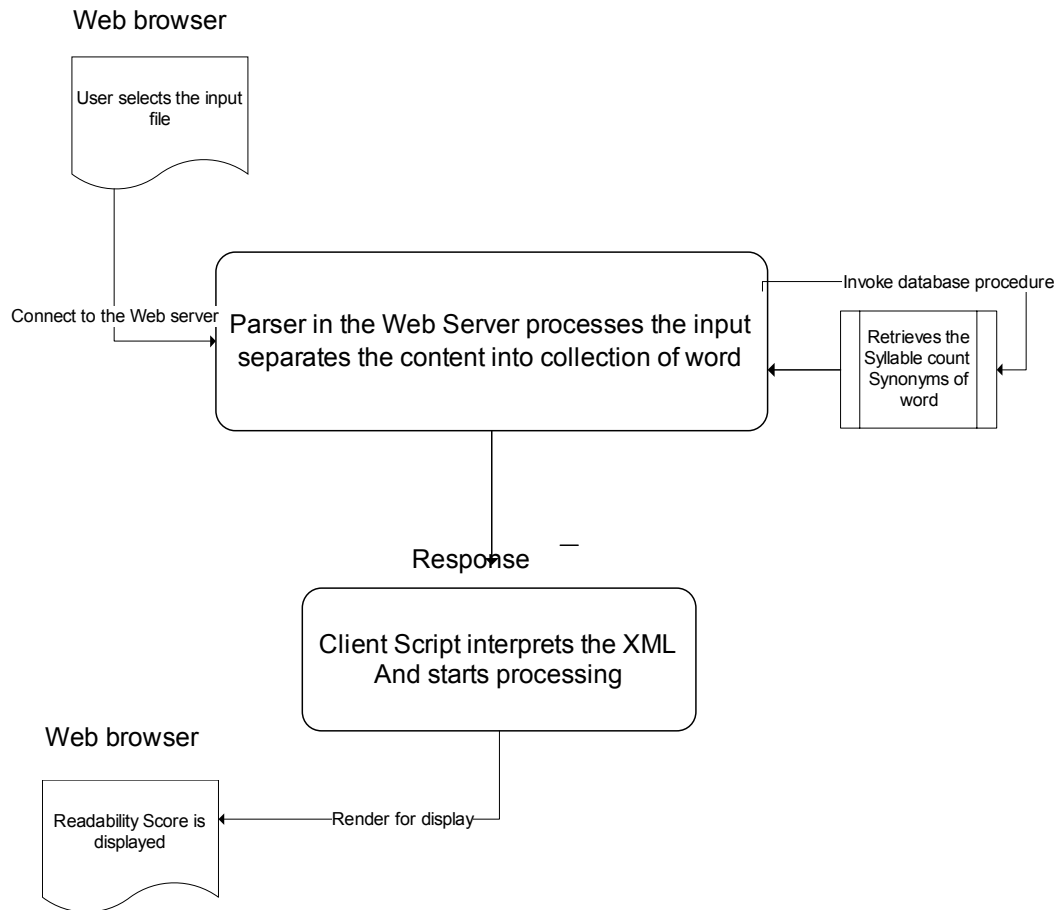


Figure 3.24 Client Script processing

As shown in the figure 3.24, the Script Engine receives the data in the XML format. The response data from server consists of a list of words, the associated syllable counts and synonyms. A script engine like VB Script runs in the client side. Currently, Script Engine can process XML data efficiently using Document Object Model (DOM). DOM offers faster navigation and parsing of XML data. For example, if the Synonyms list of a word is required, the Script will use DOM method to retrieve the data. XPath is another powerful standard that can facilitate retrieving a portion of the document.

Table 3.2 Actions performed by Client Script

Action	Distributed Application	Web Application
Parsing the input file	Server	Server
Loading the lexicon data	Server	Server
Display the text with synonym list	Server	Client Script
Calculate the readability score	Server	Client Script

As shown in the Table 3.2, In case of the distributed application the Parser running in the server is responsible for calculating the readability score. In the Web application, the Client Script performs this operation.

3.5.3 Web Interface of IDR application

A Web application offers a rich Graphical User Interface. Software vendor like Microsoft has introduced a number of Web controls. These controls are readily available and easy to use. A Developer can utilize these controls to offer user a better navigation and browsing experience. In our application we used a number of Web controls to provide the user a better look and feel and an easy to use interface.

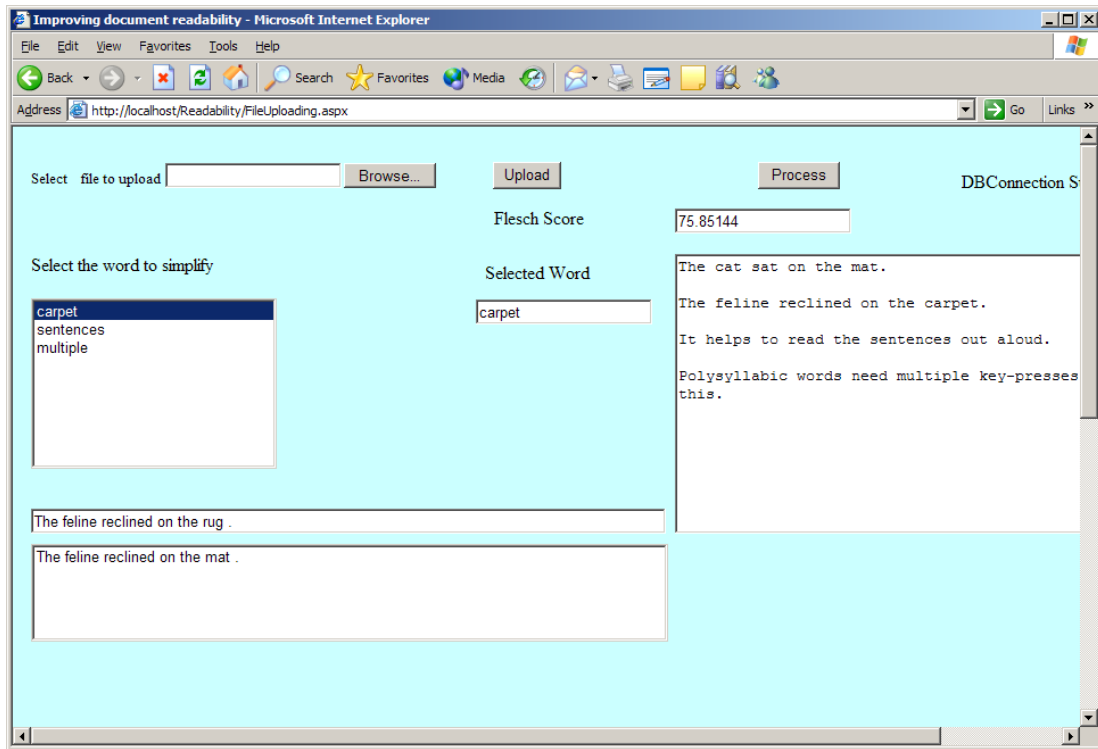


Figure 3.25 Web Interface of IDR application

There are a number of issues about deploying our Web interface. As shown in the figure 3.25, our Web interface allows the user to upload an input file to a server for processing. It is expected that a Web user will input a file of relatively small size. To ensure this we have restricted the input file size to a manageable value. If the input file size is bigger, the Script rejects the input. We are also implementing Windows integrated security, thereby forcing users to authenticate using NT security credentials. This has ensured that only valid users can access our IDR Web interface.

CHAPTER 4

CONCLUSIONS AND FUTURE WORK

4.1 Conclusions

The readability of a document can be improved with human involvement if the system provides better alternatives to restructure the text. It is also possible to provide this system to a group of users having specific requirement. This observation was mentioned at the beginning of our thesis.

Preliminary result in the project suggested that the semi-automated system that we developed assists user to improve the readability of the documents. In addition to improving, extending and evaluating our text simplification system, we also wanted to evaluate whether we can provide the application to a wider audience. We developed a number of prototypes for Windows stand-alone, Client Server, Software Agent and Web environments. There were a number of issues that we had to solve. However, because of the advancement in technology we were able to provide an effective solution and also ensure that the system is efficient and user friendly. Our conclusion is that a semi automated system with user involvement can significantly improve the document readability and because of the current distributed technology it is possible to offer this solution to group of users.

4.2 Future Work

A significant source that can be used for improving document readability is the WorldNet [Mill 97]. Like the lexicon, WorldNet also contains the words and related information. However, WorldNet also includes the part of speech of a word. Moreover, a WorldNet thesaurus consists of synonyms groups. The thesaurus divides the words into groups of synonyms. WorldNet uses the term “Synset” for synonym groups. For example, WorldNet include the words “beast”, “brute”, “creature” and “fauna” into the single “Synset” “animal”. It is possible to associate the lexicon with WorldNet database. This way user can be provided with better alternatives.

Our improving document readability tool can also be extended to interface with Microsoft Word. Microsoft Visual Basic for Application [VBA] has provided a number of functions to support operations related to Word documents. An application can use these references to access Word or Excel. The current IDR tool can process text files. Using VBA, it would be possible to allow user to access Word files.

The current trend in document readability research is to incorporate document content in measuring reading complexity. For example, Web documents, have very different characteristics than newspaper articles or pages in a textbook. In this type of document, a statistical model will provide useful information on the content of the document. For example, a normal distribution with a specific mean and variance can be used to model the sentence length distribution of each readability grade. In this way, this statistical model can be combined with readability matrices to measure the document complexity more accurately [Si Callan 05].

We feel that IDR tool that we developed can assist in this type of work. Our Web and Internet based applications have provided the necessary framework and this can be used as an effective tool for further research and future work.

APPENDIX A

EXPERIMENTAL RESULT

We have developed a prototype for each of our techniques for Improving Document Readability. We have already discussed the underlying architecture and implementation. In this appendix, we compare the performance of these design alternatives. We have used the corpus of text to evaluate the performance and we performed the test on Windows stand-alone, Client Server and Software Agent environment. The following parameters/variables were used to determine their effect on the overhead of each of the alternatives.

Data retrieval time – One of the important steps in our IDR application is the retrieval of the lexicon. Data retrieval time is the time required to load the lexicon. The current lexicon has 150 K records and because of this size, retrieval time is significant in overall performance of our application. Each technique that we implemented has employed a different strategy to retrieve data. Windows stand-alone retrieves the lexicon in a single data request. Client Server uses the concept of vertical slicing to retrieve data in slices. The Software Agent is using the XML representation of data. We evaluated the Data retrieval time in each of these implementations.

Processing time – Processing time is the time to complete the initial parsing of text. We are using the Parser object to parse the input file. However, the parsing mechanism is different in each of our implementations. In case of Windows stand-alone model this information is retrieved from the data table. However, for Software Agent this information is accessed from XML repository using Document Object Model and XPath. So, these implementations will have different processing time. This

processing time is also going to have significant impact in the performance of our application.

We have calculated the data retrieval time and processing time by using the “Date Time” Windows utility. We can use this utility to mark a specific instance in terms of hours, minutes, seconds and milliseconds. To calculate the Data retrieval time, we mark the time stamp before loading the database and the time stamp after completion of the data load. The difference shows the Data retrieval time. Similarly we also calculated the Processing time by marking the instance before and after the completion of parsing.

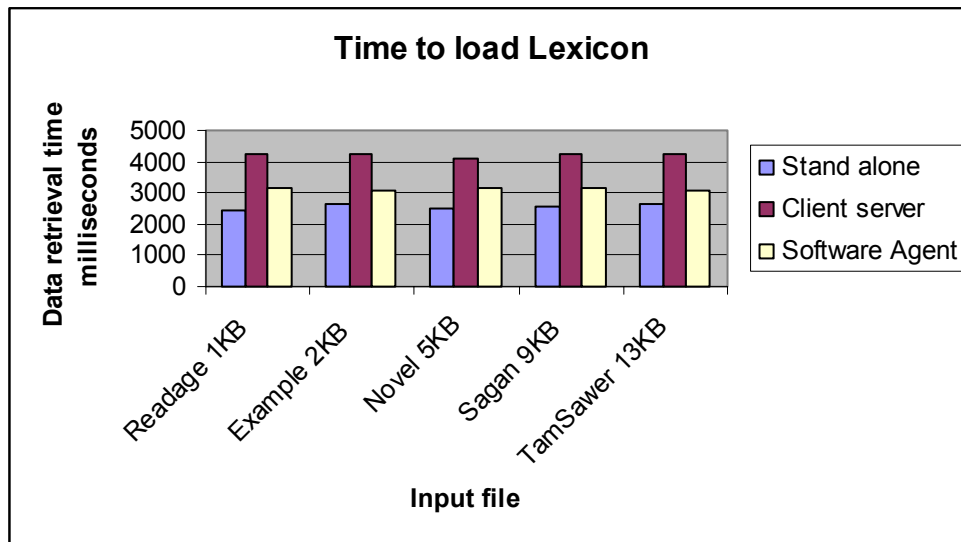


Figure A.1 Comparison of data retrieval time

We have measured the data retrieval time using a number of text files with varying size. The IDR tool loads the lexicon when a user clicks the process button.

When the user clicks the process button again to process another input file, the lexicon is already in memory and as such loading is not required. However our objective is to measure the data retrieval time with input files of different sizes. So we close the system when we are done with measuring a particular test file. We restart the application when we are ready to test the next file. This ensures that lexicon is loaded each time the system process a file. As shown in the figure A.1, The stand-alone environment has the best response; Client Server is the slowest. This can be explained by the fact that stand-alone and Software Agent lexicons reside in the client machine and as such take less time to load. In the Client server, the lexicon is located in the server and is transmitted over the network. This explains the slower response of Client Server model over the desktop or Software Agent. We also observed that data retrieval time is nearly same for files of different sizes.

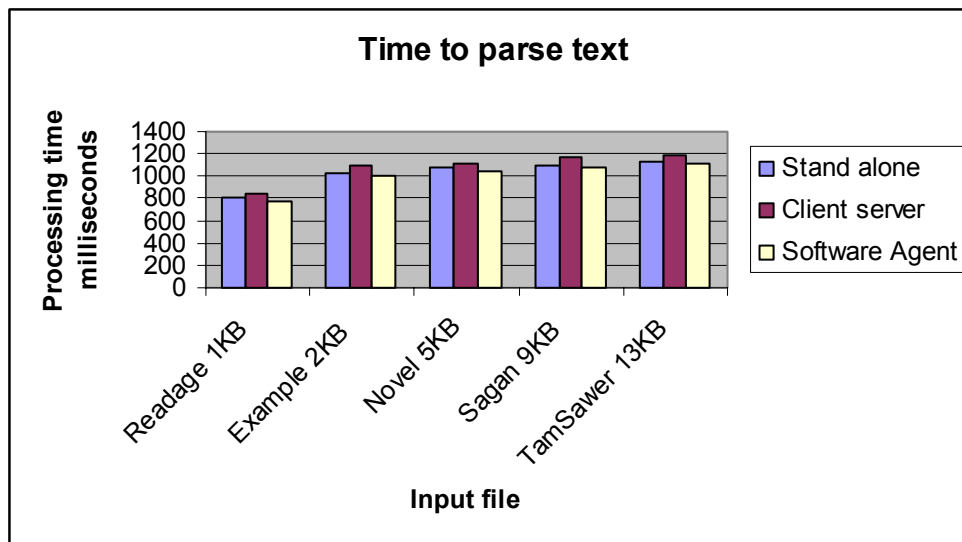


Figure A.2 Comparison of processing time

We measure the processing time of input files of different sizes. Processing time is the time required to parse the document and to calculate the syllable count of each word of the document. The syllable count is retrieved from the lexicon which is already loaded in memory. We perform the measurement several times for each of the input files. The entry in the chart is average of these values. As shown in the figure A.2 Processing time is almost identical for stand-alone and Client Server. This can be explained by the fact that the lexicon is already loaded in memory in both the environment. As such the processing time is almost same for these two environments. The processing time of Software Agent is better than the other two. In Software Agent, we retrieve the syllable count (necessary for parsing) from XML. This implies that the retrieval is faster in XML compared to relational database. However this observation may not be true for each relational database. Currently our lexicon database is Microsoft Access. We could use relational database like Oracle to compare and evaluate the performance of using XML to come to a more general conclusion.

APPENDIX B

READABILITY FIGURES FOR SAMPLE DOCUMENTS

Appendix B consists of a number of tables that show the various statistics produced by IDR on sample text files. We are also comparing this with statistics produced by other applications.

The first cell of the first column of the table is the name of the document file. The second column is the name of the software that produced the statistics related to the document. For example, Microsoft Word can display the number of words, sentences of a document. There are also a number of commercial applications that can display the statistics related to a document. These statistics are collected from the repository of Carrick [Carrick 2000] and are shown in the remaining columns.

Table B.1 Comparing IDR ‘AddCopy’ statistics

Name of document	Software	Words	Sentences	Syllables	Flesch score	Flesch grade level
AdCopy	MS-Words	536	40	868	56.30	8.70
	ProScribe	533	37	853	57.00	11.00
	RightWriter	536	41	843	60.51	8.07
	Rewrite	487	34	762	59.92	8.46
	IDR	533	42	850	60.23	8.90

Table B.2 Comparing IDR ‘EasyWriter’ statistics

Name of document	Software	Words	Sentences	Syllables	Flesch score	Flesch grade level
EasyWriter	MS-Words	101	2	154	52.20	12.00
	ProScribe	101	4	152	51.00	13.00
	RightWriter	101	4	154	52.21	12.00
	Rewrite	100	4	150	54.56	11.86
	IDR	104	5	155	59.63	13.00

Table B.3 Comparing IDR ‘ReadAge’ statistics

Name of document	Software	Words	Sentences	Syllables	Flesch score	Flesch grade level
ReadAge	MS-Words	27	4	40	74.60	4.50
	ProScribe	27	4	41	71.00	8.00
	RightWriter	27	4	41	71.52	4.96
	Rewrite	28	4	41	75.85	4.42
	IDR	28	4	41	75.85	4.80

Table B.4 Comparing IDR ‘RightWriter’ statistics

Name of document	Software	Words	Sentences	Syllables	Flesch score	Flesch grade level
RightWriter	MS-Words	75	4	125	46.80	11.30
	ProScribe	75	5	128	47.00	14.00
	RightWriter	75	4	125	46.80	11.39
	Rewrite	76	6	129	50.38	9.38
	IDR	77	6	130	49.60	10.00

We have compared the statistics of IDR with other applications. We used four test files for these comparisons. Our observation is that statistics of IDR agree with other systems except for a few anomalies. The Sentence count of IDR is higher compared with others. The IDR application is considering certain characters as the end of sentence while for other applications these characters are considered as the continuation of the sentence. This can be the explanation for higher sentence count of IDR.

APPENDIX C

READABILITY STATISTICS COMPUTED FOR TEST DOCUMENTS

Appendix C contains the documents that we used to test IDR for improving document readability. For each document, there is a revised version. The revised version is obtained using the IDR. In both the versions, we also show the readability scores.

Document Name: ReadAge

Original Version Readability Score: 75.85

The cat sat on the mat.

The feline reclined on the carpet.

It helps to read the sentences out aloud.

Polysyllabic words need multiple key-presses, like this.

Revised Version Readability Score: 78.87

The cat sat on the mat.

The feline reclined on the mat.

It helps to read the sentences out aloud.

Polysyllabic words need multiple key-presses, like this.

Document Name: Fog 296

Original Version Readability Score: 23.89

I urgently recommend maintaining in abeyance any premature implementation of the proposed sales promotional prospectus until we analyze the resultant data of the consumer motivational investigations Harris is researching. His preliminary conclusions indicate evidence of the existence of previously unconsidered parameters which eventually may prove to be counterproductive.

Revised Version Readability Score: 39.27

I urgently urge maintaining in abeyance any premature execution of the proposed sales promotional prospectus until we study the sequent data of the consumer motivational investigations Harris is researching. His preliminary conclusions argue evidence of the existence of previously unconsidered parameters which yet may prove to be counterproductive.

Document Name: Easy Writer

Original Version

Readability Score: 59.63

Then halfway down the list, I realized one important item hadn't even been included: a changing table.

That evening we both sat down and designed our own changing table (we had painted the baby's room, why not add one more personal touch?). Neither of us are skilled carpenters, but the efforts of our love was a table that was extremely economical (all materials under \$30), simple to construct (no nails needed), quickly taken apart for moving and storage, easily converted into a toy chest, and nice to look at! We've tested the table for nineteen months and can report that it.

Revised Version

Readability Score: 61.23

Then halfway down the list, I realized one important item hadn't even been included: a changing table.

That evening we both sat down and designed our own changing table (we had painted the baby's room, why not add one more personal touch?). Neither of us are skilled carpenters, but the efforts of our love was a table that was extremely economical (all materials under \$30), simple to make (no nails needed), quickly taken apart for moving and storage, easily converted into a toy chest, and nice to look at! We've tested the table for nineteen months and can report that it.

REFERENCES

- [Dale and Chall 48], Dale, E ., and Chall, J . "A Formula for Predicting Readability", Educational Research Bulletin, Volume 27, January 1948, pp 11-20.
- [Flesch 48] Flesch, R. "A New Readability Yardstick", Journal of Applied Psychology, Volume 32, June 1948, pp .221-233
- [Bormuth 66] Bormuth, J . "Readability : A New Approach," Reading Research Quarterly, Volume 1, Number 3, 1966, pp. 79-132
- [Gunning 52] Gunning, "The Technique of Clear Writing", McGraw Hill, New York, 1952
- [Klare 77] Klare, G. "Readable Technical Writing: Some Observations", Technical Communication, Volume 24, Number 2, 1977, pp. 1-5.
- [Klare 55] Klare, G, Mabry, J , and Gustafson, L . "The Relationship of Style Difficulty to Immediate Retention and to Acceptability of Technical Material," Journal of Educational Psychology, Volume 46, 1955a, pp. 287-295.
- [Klare Smart 73] Klare, G , and Smart, K . "Analysis of the Readability Level of Selected USAFI Instructional Materials," Journal of Educational Research, Volume 67, Number 4 ,1973, p . 176 .
- [Carrick 90] Albert Gilmore Carrick, "Automation of Improving Document Readability", University of Texas at Arlington, 2000
- [Samuels Zakaluk 88] Beverly L. Zakaluk and S. Jay Samuels. Readability: It's Past, Present, & Future published by the International Reading Association, Newark, 1988

[Mill 97] George A Miller, WordNet – a Lexical database for English, (Princeton, NJ Cognitive Science Laboratory, Princeton University, 1997); available from <http://www.cogsci.princeton.edu/-wn/w3wn.html/>

[OxUP63] The shorter Oxford English Dictionary(Oxford, England: Oxford University Press) , available from <ftp://ftp.white.toronto.edu/pub/words/sodict.gz>

[Ward 96] Gary Ward, The Moby lexicon project(Arcata, Calif.: Grandy Ward, 1996)

[ASP .NET 06] Zak Ruvalcaba Build Your Own ASP.NET Website using c# & VB.NET Site point , 2006.

[Pro NET 05] Matthew MacDonald and Mario Szpuszta , Pro ASP.NET 2.0 in C# Apress, Paperback, Published September 2005.

[Bentley 05] Whitten Bentley. System Analysis and Design Methods, Purdue University 2005

[Si Callan 05] Luo Si and Jamie Callan. “A Statistical Model for Scientific Readability”,School of Computer Science Carnegie Mellon, 2005.

[Lhotka 04] Rockford Lhotka. Expert c# Business Objects. Apress, Paperback, Published September 2004

BIOGRAPHICAL INFORMATION

Ankur Bora was born in Nagaon, Assam, India. He obtained his B.E. degree in Computer Science and Engineering from the University of Dibrugarh, India. Subsequently he worked at National Informatics Center, New Delhi where he got his first exposure to large scale information systems. He has been working as a Software Consultant and worked with telecommunication client Verizon, Sprint PCS and South Western Bell. He has wide experience in object oriented analysis and design and developing middle tier systems. His interest in research in information technology brought him to University of Texas at Arlington where he obtained his MS degree in Computer Science and Engineering in 2006. His research interest includes information retrieval, text processing and web search engines.