**LDoc SRS**

**Paul Scarrone**

**Senior Project Spring 2010**

**03-05-2010**

# 1.0 Introduction

## 1.1 Purpose

This document provides an outline of the Large Document Search Engine as a whole and describes the contextual analysis engine utilized in this system. This document is intended for future developers in the following groups:

Project Managers

Software Developers

Linguistic Scientists

## 1.2 Scope

The Large Document Search Engine or LDoc is a system designed to perform contextual analysis on large documents for the purpose of comparing and cataloging documents. The process of research would be greatly enhanced if passages and works could be identified and weighted based upon similarity to other documents or keyword requests. The LDoc evaluates documents by exploiting two types of nouns; standard and proper nouns. Standard nouns are expanded using a thesaurus to extract meaning. Proper nouns are identified by removing all other sentence elements (ie: verbs, adjectives, ...). Future improvement of this technique could allow analysis of an author's whole body of work. Permitting a user to make a complex request regarding the author's view of a topic. The result of this request would provide validation and supporting evidence. Furthermore, If multiple bodies of work were compared theories could be evaluated for an entire movement.

## 1.3 Terms

### 1.3.1 Definitions

Lower Order Words - These words are the top weighted contextual items associated with any noun defined by a weighting process of passing a word into a thesaurus and then its resulting synonyms are passed back into the thesaurus. The most common terms become a word's Lower Order Words.

Document Element - These consist of Whole Document, Paragraph, Sentence, and Word and define regions for comparison testing

### 1.3.2 Abbreviations

LDoc - Large Document Search Engine

LOWords - Lower Order Words

BiWord - a word set consisting of a pair of words minus spaces and puncuation

TriWord - a word set consisting or 3 words minus spaces and puncuation

## 1.4 Reference

A high level overview of contextual analysis

http://www.unl.edu/sbehrend/html/sbsite/StudyQuestions/ContextualAnalysis.html

Java API for WordNet Database

http://lyle.smu.edu/~tspell/jaws/index.html

WordNet Dictionary Source

http://wordnet.princeton.edu/

Grady Ward's Mobi Thesaurus

http://icon.shef.ac.uk/Moby/

JGAAP Authorship Software

http://server8.mathcomp.duq.edu/jgaap/w/index.php/Main\_Page

## 1.5 Overview

This document is divided into 4 sections including:

Overall Description

Specification Requirements

Supporting information

# 2.0 Overall Description

## 2.1 Product perspective

The contextual analysis engine described in this document is intended to be included in a search engine which can store processed documents for meaningful future analysis. Examples of this include the combined works of academic journals like JSTOR or the works of an author from a source like Project Gutenberg. These documents will be stored in a database so users may look for research materials in a manner beyond full text searches. Additional interfaces would allow for context keywords to be used in existing web search engines to return and on demand analyze websites for matching materials.

## 2.2 Product Function Overview

### 2.2.1 Document Processing

The main purpose of this engine is to read a large volume of plain text and return a complex object which contains all the words of the original document in relation to its location. The location of a word is in terms of the sentence and paragraph in which it existed. Words are categorized based upon their linguistic type. The hypothesis is that the context carrying members of a document are the nouns. By identifying and expressing a document in terms of its nouns it should be possible to identify the context of the whole document.

#### 2.2.1.1 Document Structure Identification

Sentences are identified by a string of words followed by a (.)period.

Paragraphs are identified by tab preceded by a (.)period.

#### 2.2.1.2 Document Canonization

Before the document is processed for its individual parts it is canonized. This results in a string of the whole document with only single spaces between text objects, tabs, (.)periods, and CR LF characters. This process helps to eliminate words being confused with special characters.

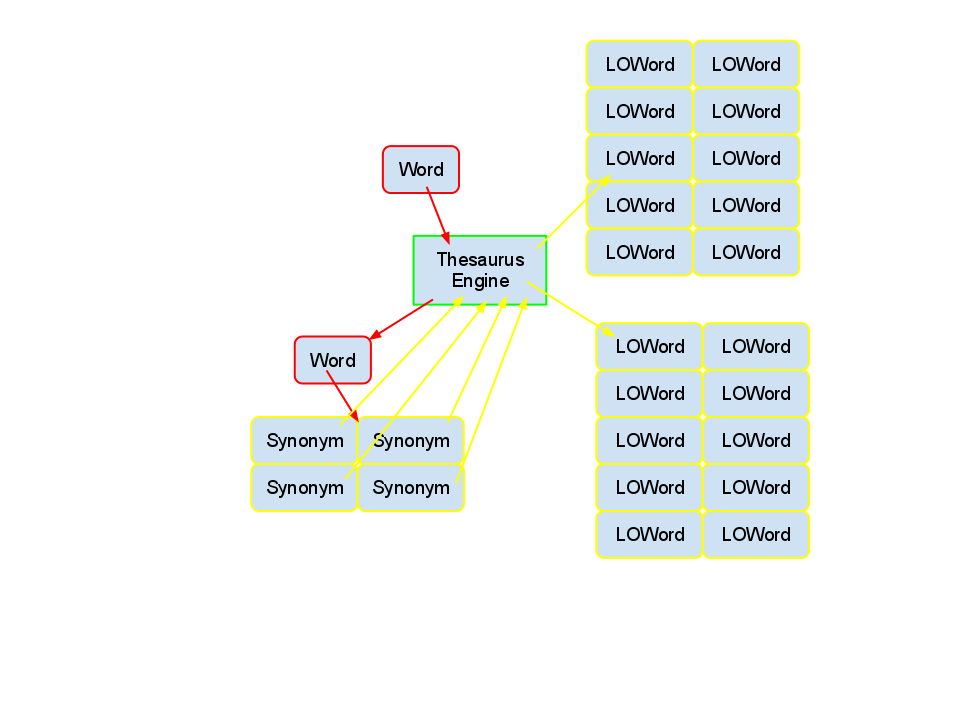
#### 2.2.1.3 Term Identification

Externally aquired dictionaries from dict.org and word.net were used to provide a means of identifying a words purpose in the sentence. Currently words are identified as Adjective, Connectives, Adverb, Verb, Noun, and Keyword. Adjectives, Connectives, Adverbs, Verbs, and Nouns are identified by comparing words to their respective dictionary lists. Keywords are the words which fall through this identification process. Keywords are expected to be proper nouns and abstract or foreign words. Keywords are the primary level of contextual comparison as they are the most unique type of words in a document. The secondary comparison items are the nouns and their weighted LOWord values. These are stored along with the word in relation to their location in a Vector. This Vector is a stacked series of objects.

#### 2.2.1.4 Term Weighting

The process of term weighting occurs in both the case of a LOWord or a keyword. LOWords are weighted a number of times before they are even affixed to their respective word. To generate the LOWords for a particular noun the noun is first compared against a thesaurus which returns a long series of synonyms. To consider the best context this list is individually compared against the thesaurus and the resulting aggregated list of words is tested for the most commonly occurring words. These most common synonyms are the LOWords of the noun. This list is limited to 5 or 10 items uniformly accross the document. At each junction where a sentence, paragraph, or the document ends the aggregated LOWords for that section are once again tested for commonality and the top set is passed on to represent that document element.

Keywords are tested for commonality the same way as nouns are with the caveat that they are not passed into the thesaurus at any time. So at each junction of document elements the keywords themselves are tested for commonality and the top most common values are assigned to that element.



### 2.2.2 Document Comparison

The truly experimental process of this engine is in how to evaluate the data between the objects. Generally when comparing two documents each element is tested against all of the same elements in another document these results are evaluated based upon the aggregation of all of the test results.

Eg. when paragraphs are tested against each other a sample output would be this:

Document Comparison = 100.00%

Paragraph Comparison = 60.00%

Sentence Comparison = 52.86%

Paragraph Flat Comparison

The Quality for Comparison of Paragraphs 0 to 0 is 100.0%

The Quality for Comparison of Paragraphs 0 to 1 is 19.999999999999996%

The Quality for Comparison of Paragraphs 0 to 2 is 40.0%

The Quality for Comparison of Paragraphs 0 to 3 is 40.0%

The Quality for Comparison of Paragraphs 0 to 4 is 0.0%

The Quality for Comparison of Paragraphs 0 to 5 is 0.0%

The Quality for Comparison of Paragraphs 0 to 6 is 40.0%

The Quality for Comparison of Paragraphs 0 to 7 is 40.0%

The Quality for Comparison of Paragraphs 0 to 8 is 0.0%

The Quality for Comparison of Paragraphs 1 to 0 is 19.999999999999996%

The Quality for Comparison of Paragraphs 1 to 1 is 100.0%

The Quality for Comparison of Paragraphs 1 to 2 is 40.0%

The Quality for Comparison of Paragraphs 1 to 3 is 0.0%

.

.

.

The Quality for Comparison of Paragraphs 6 to 5 is 0.0%

The Quality for Comparison of Paragraphs 6 to 6 is 100.0%

The Quality for Comparison of Paragraphs 6 to 7 is 40.0%

The Quality for Comparison of Paragraphs 6 to 8 is 0.0%

The Quality for Comparison of Paragraphs 7 to 0 is 40.0%

The Quality for Comparison of Paragraphs 7 to 1 is 0.0%

The Quality for Comparison of Paragraphs 7 to 2 is 0.0%

The Quality for Comparison of Paragraphs 7 to 3 is 40.0%

The Quality for Comparison of Paragraphs 7 to 4 is 0.0%

The Quality for Comparison of Paragraphs 7 to 5 is 40.0%

The Quality for Comparison of Paragraphs 7 to 6 is 40.0%

The Quality for Comparison of Paragraphs 7 to 7 is 100.0%

The Quality for Comparison of Paragraphs 7 to 8 is 0.0%

The Quality for Comparison of Paragraphs 8 to 0 is 0.0%

The Quality for Comparison of Paragraphs 8 to 1 is 0.0%

The Quality for Comparison of Paragraphs 8 to 2 is 0.0%

The Quality for Comparison of Paragraphs 8 to 3 is 0.0%

The Quality for Comparison of Paragraphs 8 to 4 is 19.999999999999996%

The Quality for Comparison of Paragraphs 8 to 5 is 40.0%

The Quality for Comparison of Paragraphs 8 to 6 is 0.0%

The Quality for Comparison of Paragraphs 8 to 7 is 0.0%

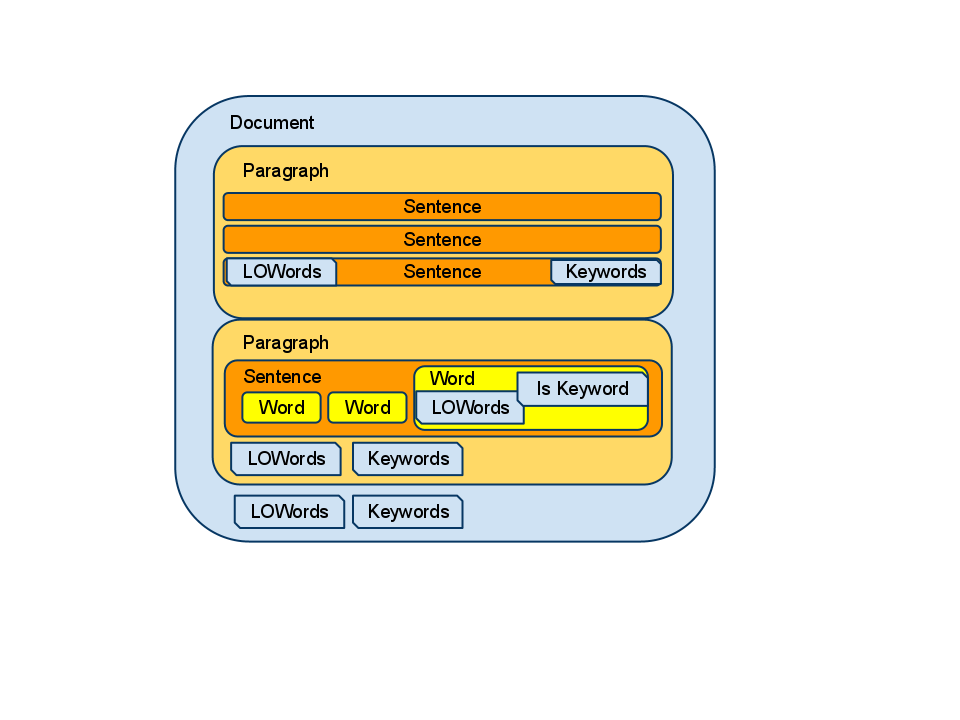
The Quality for Comparison of Paragraphs 8 to 8 is 100.0%

0.23950617283950618

These documents are in fact identical. This is evident if we look at the flat document comparison and see that when the paragraph numbers match the quality is 100% While other paragrahs match partially. Currently identical document all score approximately 24% on a flat comparison with reasonable consistency no matter the number of paragraphs.

### 2.2.3 Available Data Objects

#### 2.2.3.1 Processed Document



#### 2.2.3.2 Reconstructed Document

For additional testing the processed document is also converted into a flat array of objects which instead of being nested by document element contain an ID number for the elements they belong to. The reconstructed document is also used to reintegrate the document so word distances need to be calculated.

## 2.3 User Characteristics

As a standalone engine for comparison testing users would required general programming skills and experience with Java programming in Eclipse to make positive use of this system.

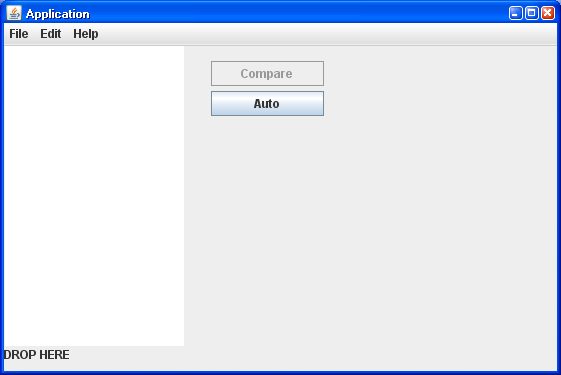
## 2.4 Constraints

The application is restricted to running in Eclipse for testing and reporting in this developmental phase and requires that arguments be set for Java to properly execute document comparisons. To avoid out of memory errors use the arguments -Xms1024M -Xmx1024M or greater values.

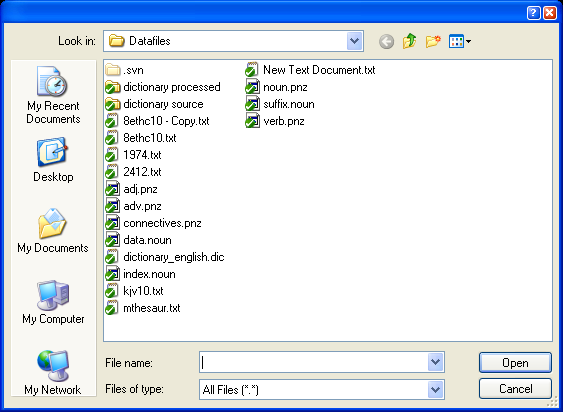
# 3.0 Specification Requirements

## 3.1 External Interfaces

Gui is currently for testing comparison only and will change when user comparisons are implemented. The main window allows for documents to be loaded for processing using a dropdown File->Open option.



The open dialog is the OS default dialog and is cross platform compatible.



## 3.2 Classes

The program has two entry points. First, is the App.java class which is the driver to handle creation of the processed document objects. Second, is the MainApplication.java class which generates a GUI for testing documents and concurrent document processing and calls the App.java driver for each file loaded.

### 3.2.1 MainApplication

MainApplication handles the generation of the main testing GUI and handles concurrent file processing by threading each new File->Open and Comparison separately.

#### CRC

Properties

logger : Logger

application : MainApplication

listModel : DefaultListModel

selected : Object[]

jFrame : JFrame

jContentPane : JPanel

jJMenuBar : JMenuBar

fileMenu : JMenu

editMenu : JMenu

helpMenu : JMenu

exitMenuItem : JMenuItem

openMenuItem : JMenuItem

aboutMenuItem : JMenuItem

cutMenuItem : JMenuItem

copyMenuItem : JMenuItem

pasteMenuItem : JMenuItem

saveMenuItem : JMenuItem

aboutDialog : JDialog

aboutContentPane : JPanel

aboutVersionLabel : JLabel

jTextPane : JTextPane

jTextPane1 : JTextPane

jLabel : JLabel

analyzingObjects : Vector<App>

analyzedObjects : Vector<App>

jList : JList

jButton : JButton

Methods:

getJTextPane()

getJTextPane1()

updateFileList()

getJList()

getJList().new ListSelectionListener() {...}

getJButton()

getJButton().new ActionListener() {...}

main(String[])

main(...).new Runnable() {...}

main(...).new Runnable() {...}

getJFrame()

getJContentPane()

getJJMenuBar()

getFileMenu()

getOpenMenuItem()

getOpenMenuItem().new ActionListener() {...}

getEditMenu()

getHelpMenu()

getExitMenuItem()

getExitMenuItem().new ActionListener() {...}

getAboutMenuItem()

getAboutMenuItem().new ActionListener() {...}

getAboutDialog()

getAboutContentPane()

getAboutVersionLabel()

getCutMenuItem()

getCopyMenuItem()

getPasteMenuItem()

getSaveMenuItem()

### 3.2.2 App

App is the main driver for all document processing. Not only does negotiate the reading of the document to be processed but also creates the nested document object and the flattened sequence object. Please refer to the minimalist sequence diagram. It excludes trivial functions like the logger from the classes description.

#### CRC

Properties:

\_done : boolean

logger : Logger

Filename : String

EvaluateSets : WordWorkingSets

file : StripCase

SequencedDocument : ArrayList<ReconstructDocument>

ProcessedDocument : Document

threadID : int

FullFileName : String

Methods:

getProcessedDocument()

App()

App(String)

getSequencedDocument()

isDone()

getFullFileName()

getFileName()

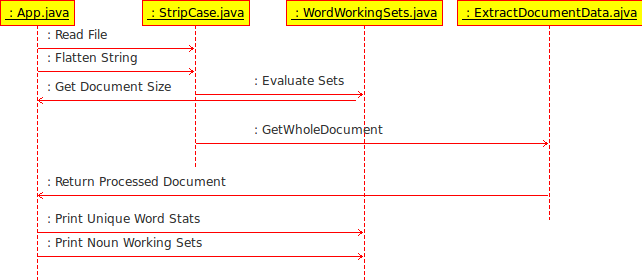
setThreadID(int)

getThreadID()

setisDone(boolean)

ProcessFile(String)

#### Sequence Diagram

****

#### Figure 3.2.2.1

### 3.2.3 ExtractDocumentData

By utilizing the WordManagement class to provide word lists this class breaks the entire document into its core components, Words. It sequentially identifies each word by type and handles term weighting on nouns and keywords. strings are initially delimited by (.)period and stored as sentences followed by tab delimiting for paragraph identification. This information is stored as a nested Document object as described in section 2.2.3.1

#### CRC

Properties:

loggerDocument : Logger

loggerParagraph : Logger

loggerSentence : Logger

FinalDocument : Document

WholeDocument : String

NounList : WordManagement

ConnectivesList : WordManagement

AdverbList : WordManagement

AdjectiveList : WordManagement

VerbList : WordManagement

DocSize : double

CurrentWord : double

Methods:

ExtractDocumentData()

ExtractDocumentData(String, double)

setDocSize(double)

getFinalDocument()

setWholeDocument(String)

BreakOnDocument()

### 3.2.4 SequenceDocument

This class does a reversal of the work done in the ExtractDocumentData class except this storage is flat and word location is identified by IDs for each paragraph, sentence, and word. This is accomplished by traversal of the previously created object.

#### CRC

Property:

SequencedDocument : ArrayList<ReconstructDocument>

Methods:

getSequencedDocument()

SequenceDocument(Document)

### 3.2.5 StripCase

StripCase is actually more multifunction then its title would let on. This class is responsible for reading the file provided to App.java and creates a string that is all lowercase with punctuation limited to the (.)period and formatting limited to CLRF and tabs for identifying paragraphs. White space is also normalized so that no non tab whitespace is followed by additional whitespace. This eliminates null words being identified later by ExtractDocumentData.

#### CRC

Properties:

logger : Logger

WholeDocument : String

NormalizeWhitespaceFlag : boolean

Methods:

StripCase()

GetWholeDocument()

FlattenString()

NormalizeWhitespace(boolean)

ReadFile(String)

### 3.2.6 WordManagement

A general class to handle dict.org format wordlists and place them into a String Key null contents hashmap for rapid comparison to the tests in ExtractDocumentData. The wordlist format is a single word per line with no spaces or special characters. These lists are processed ahead of time and handle in a static manner.

#### CRC

Properties:

WordList : HashMap<String, Object>

file : File

Methods:

WordManagement(String)

LoadList(String)

isInList(String)

### 3.2.7 WordWorkingSets

This class handles some statistical needs for optimizing ExtractDocumentData and to provide progress feedback for the user when processing very large files.

The generated statistics are:

Number of words in document

Number of unique words in the document

Number of nouns in the document

Number of unique nouns in the document.

Since the number of nouns in a document is about 10% of the total unique words the identification process is optimized by preprocessing the LOWord terms and assigning them to their respective nouns as opposed to processing the LOWords each time the noun occurs.

#### CRC

Properties:

uniquesize : int

start : int

stop : int

WholeDocumentSize : double

wholedoc : String

uniqueworkingSet : Vector<String>

nounworkingSet : Vector<String>

Methods:

getNounWorkinSet()

getUniqueSize()

setThreadStartStop(int, int)

WordWorkingSets(String)

getDocumentSize()

BuildNounWorkingSet()

BuildNounWorkingSet(int, int)

BuildUniqueWordWorkingSet()

PrintNounWorkingSet()

PrintUniqueWordStats()

### 3.2.8 ThesaurusHandler

In a similar way this class acts very similarly to the WordManagement class except the format of the files it handles is different. The thesaurus used is Grady's Moby Thesaurus. The format of this document is a word on each line followed by its synonyms comma delimited. Unlike WordManagement the hashmap created uses the first word on a line as the key value and places its synonyms in an arraylist. This class also handles reducing the list of synonyms returned from the thesaurus lookup to a constant length.

#### CRC

Properties:

file : File

Thesarus : HashMap<String, ArrayList<String>>

LOWordSize : int

Methods:

ThesaurusHandler()

ThesaurusLookup(String)

ReduceSynonyms(Vector<String>)

CorrelateThesaurusItems(String)

### 3.2.9 WordListTools

This is a general purpose static class with functions derived from the WordManagement method which identifies the most commonly occurring items in a list of words. Primarily used to handle the keyword filtering that occurs at each junction of a document element in ExtractDocumentData.

#### CRC

Properties:

ListLimit : int

Methods:

WordListTools(int)

TopItems(Vector<String>, int)

TopItems(Vector<String>)

### 3.2.10 CompareDocuments

This is a temporary class for testing methods of evaluating the results of two or more processed files. When fluent methods of identifying documents is discovered this class will be depricated in lue of individual classes which will handle each comparison method.

#### CRC

Properties:

Selection : ArrayList<SelectionItems>

logger : Logger

AnaylzedDocuments : Vector<App>

Comparables : ArrayList<Document>

DocumentComparison : ArrayList<Comparisons>

Methods:

CompareDocuments(ArrayList<SelectionItems>, Vector<App>)

compareDocuments()

CmpLOWords(Vector<String>, Vector<String>)

LOWordComparison(Vector<Vector<String>>)

gatherParagraphsLOWords()

gatherSentencesLOWords()

retrieveDocumentLOWords(Document)

retrieveParagraphLOWords(Paragraphs)

retrieveSentenceLOWords(Sentences)

retrieveWordLOWords(Document, int, int, int)

extractComparisonList()

### 3.2.11 Analysis Loader

To permit interchangeable use of analysis methods within the analysis portions of the application workflow. By using a configuration file analysis classes can be added and subtracted from the workflow dynamically.

#### CRC

Properties:

logger : Logger

TestDocuments : ArrayList<Comparables>

DocumentComparison : ArrayList<Comparisons>

Methods:

init(ArrayList<Comparables>, String[])

runAnalysis()

CmpWords(Collection<String>, Collection<String>)

FlatComparison(Collection<Vector<String>>, Collection<Vector<String>>)

KeywordComparison(Collection<Vector<String>>)

gatherParagraphsKeywords()

gatherSentencesKeywords()

retrieveDocumenKeywords(Document)

retrieveParagraphKeywords(Paragraphs)

retrieveSentenceKeywords(Sentences)

### 3.2.12 ComponentIface

Interface class to act as template for all analysis engine components.

#### CRC

Properties:

TestList : ArrayList<Document>

config : String[]

Methods:

init(ArrayList<Comparables>, String[])

runAnalysis()

### 3.2.13 SynonymAnalysis

Analysis is performed against LOWord pairs across the Document, Paragraph, and Sentence levels. LOWords are gathered from like elements of both documents and compared for frequency. The more closely the LOWords match the better the result of the analysis.

#### CRC

Properties:

logger : Logger

TestDocuments : ArrayList<Comparables>

DocumentComparison : ArrayList<Comparisons>

Methods:

init(ArrayList<Comparables>, String[])

runAnalysis()

CmpLOWords(Collection<String>, Collection<String>)

FlatComparison(Collection<Vector<String>>, Collection<Vector<String>>)

LOWordComparison(Collection<Vector<String>>)

gatherParagraphsLOWords()

gatherSentencesLOWords()

retrieveDocumentLOWords(Document)

retrieveParagraphLOWords(Paragraphs)

retrieveSentenceLOWords(Sentences)

retrieveWordLOWords(Document, int, int, int)

### 3.2.14 PhraseAnalysis

Phrase analysis takes elements in sequence from the processed document and put them into BiWord and TriWord sets and then compares the sets against another document. The closer the sets match the higher the comparison score. Sets consist of either 2 or 3 words without spaces or punctuation between them.

#### CRC

Properties:

TestList : ArrayList<Comparables>

config : String[]

TestResults : ArrayList<PhraseAnaysisResults>

logger : Logger

Methods:

init(ArrayList<Comparables>, String[])

runAnalysis()

compareTriWords()

compareBiWords()

generateBiWords(ArrayList<ReconstructDocument>)

generateNounBiWords(ArrayList<ReconstructDocument>)

generateTriWords(ArrayList<ReconstructDocument>)

### 3.2.15 BayesianAnalysis

Bayesian analysis implements a very naive analysis engine which computes the probability that the LOWords in on document exist in another document. Document processing order plays a major role in the calculated output. While the value returned from this analysis tends to accurately describe document similarity it has yet to prove itself a method of identify contextual similarity.

#### CRC

Properties:

TestList : ArrayList<Comparables>

config : String[]

TestResults : ArrayList<PhraseAnaysisResults>

logger : Logger

Methods:

init(ArrayList<Comparables>, String[])

doStuff(Comparables, Comparables)

runAnalysis()

## 3.3 Software System Attributes

### 3.4.1 Availability

As an end user research platform the only factor defining system availability is the amount of system memory and the presence of the Java runtime. This application requires a heap size of at least 1024 for both the main process and spawned threads.

### 3.4.2 Security

This application does not currently allow for any privileged access to the system which hosts the applications process.

### 3.4.3 Maintainability

Modification to the runtime of the document processor is housed within a single Class allowing for future changes to each document process event to be introduced with minimal modification. Document objects are kept at the top level of the end user runtime and can be passed from the gui or the main document processor without conflict with the rest of the system.

### 3.4.4 Portability

LDoc is completely cross platform under the standards that Java provides for its runtime to exist in multiple platforms. The only system specific issues are CR LF items in document encoding, directory separators, and OS specific dialogues which are handled natively by Java. The system has been tested to be fully portable across Windows and Linux builds which contain a current Java runtime.

## 3.4 Specific Requirements

### 3.4.1 Objects

#### 3.4.1.1 Document

This object contains a complex nested series of objects as described in section 2.2.3.1 and is available from App.java, MainApplication.java and ExtractDocumentData.java. This is the primary object used for testing and includes all of the documents words which have been identified by the document processor along with LOWords and keywords.

This is an example of the Document object hierarchy and associated properties.

public class Document

{

public Vector<String> topLOWords = new Vector<String> ();

public Vector<String> topKeywords = new Vector<String> ();

public Document(){}

public Vector<Paragraphs> Block = new Vector<Paragraphs> ();

{

public Vector<String> topLOWords = new Vector<String> ();

public Vector<String> topKeywords = new Vector<String> ();

public int length;

public Vector<Sentences> Paragraph = new Vector<Sentences> ();

{

public Vector<Words> Sentence = new Vector<Words> ();

public Vector<String> topLOWords = new Vector<String> ();

public Vector<String> topKeywords = new Vector<String> ();

public int length;

public boolean contextSentence;

public Vector<Words> Sentence = new Vector<Words> ();

{

public String Word;

public Vector<String> LOWords = new Vector<String> ();

public int length;

public boolean isKeyword;

public String WordType;

public Words(){}

}

}

}

}

#### 3.4.1.2 Sequenced Document

The Sequenced document is a flattened version as described in Section 2.2.3.2 and lacks the identification information for each word but maintains the words position in the document for distance calculation and document regeneration.

Each element of this object is as such:

public class ReconstructDocument

{

public String wordcontent;

public int paragraphID;

public int SentenceID;

public int wordID;

public int wordLength;

}

#### 3.4.1.3 Selection Items

This class defined structure is used to pass completed document objects from the MainApplication.java GUI to the associated comparison classes. This is used in conjunction with the analyzed documents collection which is defined by all of the App.java objects containing the objects from Sections 3.5.3.1 and 3.5.3.2

Object Definition:

public class SelectionItems {

public String Filename;

public int ID;

}

### 3.4.2 Feature

Current user interaction is limited while testing the contextual comparison results. Three primary elements exist with which a user can interact.

#### 3.4.2.1 File->Open Dialog

This event draws the OS defined open dialog and passes the resulting value to App.java for processing.

##### Event Handler

From MainApplication.java

private JMenuItem getOpenMenuItem() {

if (openMenuItem == null) {

openMenuItem = new JMenuItem();

openMenuItem.setText("Open");

openMenuItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

FileDialog fd = new FileDialog(jFrame, null, FileDialog.LOAD);

fd.setLocation(50, 50);

fd.setVisible(true);

final FileDialog fd2 = fd;

new Thread(new Runnable() {

@Override

public void run() {

try {

logger.trace("File Dialog: "+fd2.getDirectory() + fd2.getFile());

App element = new App(fd2.getDirectory() + fd2.getFile());

analyzingObjects.add(element);

}catch (Exception e) {

logger.trace("Cannot Create New Thread");

}

}

}).start();

}

});

}

return openMenuItem;

}

#### 3.4.2.2 Compare Button

When two or more items in the processed document list box are highlighted this button is accessible to the user. It fires the comparedocuments process which evaluates the selection for contextual similarity using a multitude of experimental tests and weights.

##### Event Handler

private JButton getJButton() {

if (jButton == null) {

jButton = new JButton();

jButton.setSize(113, 25);

jButton.setLocation(new Point(207, 15));

jButton.setText("Compare");

jButton.setEnabled(false);

jButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

new Thread(new Runnable() {

private SelectionItems SelItems = null;

@Override

public void run() {

String filename = "";

ArrayList<SelectionItems> Selection = new ArrayList<SelectionItems>();

for(Object item : selected) {

String[] splititem = item.toString().split(" ");

for(int i = 2; i<splititem.length;i++) {

filename = filename+ " " +splititem[i];

}

filename = filename.substring(1);

SelItems = new SelectionItems();

SelItems.ID = Integer.parseInt(splititem[1]);

SelItems.Filename = filename;

Selection.add(SelItems);

filename = "";

}

@SuppressWarnings("unused")

CompareDocuments Comparison = new CompareDocuments(Selection, analyzedObjects);

}

}).start();

}

});

}

return jButton;

}

#### 3.4.2.3 Processed Documents List

This control is shown to the user as a multi-selection file list. Each successfully processed document is placed in this list and can be highlighted by the user. When two or more objects are highlighted this control activates the compare button so that the selected objects can be passed to the CompareDocuments.java class. Selections are passed into a list which identifies the filename and the file ID of each selection so that only the required items are transmitted to CompareDocuments.java.

##### Event Handler

private JList getJList() {

if (jList == null) {

jList = new JList();

jList.setBounds(new Rectangle(1, 0, 179, 300));

jList.addListSelectionListener(new ListSelectionListener() {

public void valueChanged(ListSelectionEvent evt) {

if (!evt.getValueIsAdjusting())

{

JList list = (JList) evt.getSource();

selected = list.getSelectedValues();

if (selected.length >= 2){

jButton.setEnabled(true);

}

else {

jButton.setEnabled(false);

}

}

}

});

}

return jList;

}

### 3.4.3 Response

System responses are logged for each document processed by Log4j in the following files:

CompareDocumentThread.txt:

Shows the results of the document comparison for the last comparison

DataProcessThread.txt

Shows the term identification for the last document processed

Document.txt

Shows the words and LOWords of the last document processed for each document

Paragraph.txt

Shows the words and LOWords of the last document processed for each paragraph

Sentence.txt

Shows the words and LOWords of the last document processed for each sentence

## 3.5 Additional Comments

This engine is mainly to prove the theory of Nouns as a source of contextual meaning for written documents.

### System Limitations:

LDoc is a rather sizeable project, whose scope is not simply limited to that of design and engineering. In order to complete a contextual analysis engine the methods by which to analyze a document had to be designed. The base theory is that the meaning of a document can be extracted from nouns. The frequency and type of nouns provide a fingerprint for a document. With this fingerprint it is possible to see which other documents and sections of documents are similar. The process of extracting noun data from a document has some rather critical external requirements. First, all punctuation and excess spaces must be removed from a document. Additionally character encoding has a major effect on the software's ability to process a document correctly. Java's Scanner library is very well versed at processing character data from multiple types of encoding but still has definite issues with poorly encoded files. In some cases standard documents with end of file characters trapped in the encoding will cause poor or no processing to occur. I have not been able to identify the characters that cause Scanner to function improperly but the solution to this problem will be to read the characters from the file directly and filter out special characters before passing it on to the tokenizer. Since this is a infrequent problem that is in reality external to the scope of the application and can be handled by transposing the text through a proper text editor I have excluded a resolution to the Scanner library at this time. Once data extraction starts the system is reliant upon a series of .dict style dictionaries that have been processed into word lists. Word.net from Princeton University provides this free dictionary source to act as a lexiconical database of the human language under the BSD license. Only the words specifically listed within these wordlists are available to the processor. This unfortunately eliminates words that contain suffixes thus only words deemed standard nouns are eligible for evaluation. While this does eliminate a subset of the total words within the target classification of a document it is a universal problem and for testing the validity of the system it was excluded from the initial design.

Only Nouns are eligible for processing as LOWords, since the LOWord creation process requires the use of synonyms an additional thesaurus source was required. The thesaurus selected was Grady's Mobiword Thesaurus. This has turned out to be a far more complete thesaurus then that of the general synsets from WordNet but it lacks the word coverage of WordNet. As a result between 10% and 22% of all identifiable nouns are not eligible for LOWords. The best solution for this would be to build a caching database of thesaurus responses from Dict.org's online thesaurus. This solution is also preferable for identifying word types, but has the initial drawback of slowing down large analysis until the cached database has sufficient entries.

Due to these factors the results of document analysis have a certain amount of expected deviation. Extraneous to this deviation is the need to devise methods of comparing the extracted document data against another document. The methods currently in practice to validate Nouns as a source of meaning are:

Keyword Analysis

Synonym(LOWord) Analysis

Phrase Analysis

Naive Bayesian Filter

While these mechanisms are built upon the analysis system it should still be possible to prove the validity of the comparison techniques since the same level of deviation is available for each document compared. Unfortunately there was not enough time to properly identify which methods of analysis were most reliable and if the tests were completely valid. I have evaluated the tests by comparing documents on similar and opposing topics, as well as the same document with syntactical and semantic deviations. In most cases the analysis system has difficulty determining the difference between different contexts and writing styles. There have been particular cases where when comparing paragraphs about apples and oranges similarities arose in context . Documents from the same standardized section of the same website about their respective fruit showed high levels of similarity. This may be an artifact of the analysis being biased towards authorship which was the chronological fore-father to this system. Additionally it is not uncommon to see a disparity between documents that should return higher similarity values of meaning. In many cases the simplest solution of comparing keywords shows the greatest similarity between documents.

While contextual analysis is not a new focus for software, commercial semantic analysis is not as pervasive due to its resource usage and lack of true commercial usage outside of AI research. The biggest lessons I learned with this project were to not adhere to an unproven theory for the full lifecycle of development if not required. For example in the course of developing this software defining the LOWords for sentence elements other than Nouns was not used. As a method to test the theory behind this method of analysis it would have been useful to compare the results of the whole documents LOWords and just the Noun LOWords for their level of similarity. Additionally some analysis methods dealing with phrase analysis showed promise but lacked the addition of weighting phrases by distance along with frequency. Finally, the system is generally difficult to test, in order to evaluate an analysis method it is necessary to have test cases that can isolate the information to be validated. In many cases similarities between documents can arise without the documents being similar due to similarities in writing style. The best way to eliminate these false positives would be to use a system that can identify authorship similarities alongside the contextual similarities. Software like JGAAP, which is directly focused on identifying authorship in dissimilar documents, could prove useful in validating results. Additional systems would need to be defined to divide the document into its constituent parts via grammatical analysis and eliminate the reliance on wordlists.

Effectively LDoc is not a functional end product used to perform real life analysis but, a test bed for discovering the methods that could be used and in what conjunction to achieve the goal of contextual analysis. Future work would include additional analysis mechanisms which further explore words both under the immediate direction of Noun based analysis and more generic document analysis using additional word sources and techniques. Its modular java architecture makes it suitable for testing on a variety of platforms and environments.

# 4.0 Supporting Information

## 4.1 Table of Contents and Index

Contents

[1.0 Introduction 1](#_Toc260645469)

[1.1 Purpose 1](#_Toc260645470)

[1.2 Scope 1](#_Toc260645471)

[1.3 Terms 1](#_Toc260645472)

[1.3.1 Definitions 1](#_Toc260645473)

[1.3.2 Abbreviations 1](#_Toc260645474)

[1.4 Reference 2](#_Toc260645475)

[1.5 Overview 2](#_Toc260645476)

[2.0 Overall Description 2](#_Toc260645477)

[2.1 Product perspective 2](#_Toc260645478)

[2.2 Product Function Overview 2](#_Toc260645479)

[2.2.1 Document Processing 2](#_Toc260645480)

[2.2.2 Document Comparison 4](#_Toc260645481)

[2.2.3 Available Data Objects 6](#_Toc260645482)

[2.3 User Characteristics 6](#_Toc260645483)

[2.4 Constraints 6](#_Toc260645484)

[3.0 Specification Requirements 7](#_Toc260645485)

[3.1 External Interfaces 7](#_Toc260645486)

[3.2 Classes 8](#_Toc260645487)

[3.2.1 MainApplication 8](#_Toc260645488)

[3.2.2 App 10](#_Toc260645489)

[3.2.3 ExtractDocumentData 11](#_Toc260645490)

[3.2.4 SequenceDocument 12](#_Toc260645491)

[3.2.5 StripCase 12](#_Toc260645492)

[3.2.6 WordManagement 12](#_Toc260645493)

[3.2.7 WordWorkingSets 13](#_Toc260645494)

[3.2.8 ThesaurusHandler 14](#_Toc260645495)

[3.2.9 WordListTools 14](#_Toc260645496)

[3.2.10 CompareDocuments 14](#_Toc260645497)

[3.2.11 Analysis Loader 15](#_Toc260645498)

[3.2.12 ComponentIface 16](#_Toc260645499)

[3.2.13 SynonymAnalysis 16](#_Toc260645500)

[3.2.14 PhraseAnalysis 16](#_Toc260645501)

[3.2.15 BayesianAnalysis 17](#_Toc260645502)

[3.3 Software System Attributes 17](#_Toc260645503)

[3.4.1 Availability 17](#_Toc260645504)

[3.4.2 Security 18](#_Toc260645505)

[3.4.3 Maintainability 18](#_Toc260645506)

[3.4.4 Portability 18](#_Toc260645507)

[3.4 Specific Requirements 18](#_Toc260645508)

[3.4.1 Objects 18](#_Toc260645509)

[3.4.2 Feature 20](#_Toc260645510)

[3.4.3 Response 22](#_Toc260645511)

[3.5 Additional Comments 23](#_Toc260645512)

[System Limitations: 23](#_Toc260645513)

[4.0 Supporting Information 25](#_Toc260645514)

[4.1 Table of Contents and Index 25](#_Toc260645515)