Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Radoslav Zápotocký

Shlukování textových dokumentů a jejich částí

Clustering of text documents and their parts

Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. Michal Kopecký, Ph.D.

Studijní program: informatika

Studijní obor: softwarové systémy

Praha 2011

Velké poděkování patří hlavně vedoucímu diplomové práce RNDr. Michalovi Kopeckému, Ph.D. za nápomocnou ruku a přínosné konzultace při psaní diplomové práce.

Název práce: Shlukování textových dokumentů a jejich částí

Autor: Radoslav Zápotocký

Katedra (ústav): Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. Michal Kopecký, Ph.D.

Abstrakt: Práce analyzuje možnosti použití vektorového modelu a shlukování aplikované na jednotlivé části dokumentu – kapitoly, odstavce a věty. Součásti práce je rovněž simulační aplikace (SimDIS), napsaná v jazyce C#, která takto upravený model implementuje a nabízí nástroje pro vizualizaci vektorů a shluků.

Klíčová slova: vektorový model, shlukování, zpracování textu, C#

Title: Clustering of text documents and their parts

Author: Radoslav Zápotocký

Department: Department of Software Engineering

Supervisor: RNDr. Michal Kopecký, Ph.D.

Abstract: This thesis analyses use of vector-space model and data clustering approaches on parts of single document – on chapters, paragraphs and sentences. A simulation application (SimDIS), written in C# programming language is also part of this thesis. The application implements the adjusted model and provides tools for visualization of vectors and clusters.

Keywords: vector-space model, clustering, text processing, C#

Table of contents

1. Introduction

The number of documents in electronic form is growing rapidly with increased number of people with access to computer and Internet. With massive number of documents it is impossible for users to read them all to find the information the want. Therefore, a number of techniques were invented to search within the collection of documents and new approaches are still under research. Most of this techniques are used mainly on collections of documents and take each document as one entity.

Some of problems appearing on document collections could be seen also on document level. For example, many documents like technical manuals are often very long, not entirely well structured and with limited search or navigation ability. Those problems could be even more visible in case the documents are to be read on electronic book readers. Reading the document at whole could be also time consuming and in case the document does not contain the relevant information it could be helpful to know approximate summary of the document without any need to read it completely.

This work focuses on possibilities of applying text retrieval techniques on individual parts of a single document for addressing above mentioned issues. We will use vector-space model to look at the document as a hierarchy of vectors, and provide tool for document content analysis.

The goal of this thesis is to create a prototype application that will allow indexing parts of the document in different levels of granularity, processing them using various text retrieval approaches and clustering them according to mutual similarity by different clustering algorithms. The application should also visualize obtained data and make them available for further analysis. Never the less, it should be designed with respect to requirement on adding new extensions and features in the future.

The rest of this thesis is structured as follows:

The second chapter provides general introduction into text retrieval topic based on vector representation of documents that allows various needed computations known from linear algebra. Later it describes transformation of classical approach to indexing and processing document parts. Following third chapter takes closer look at implementation of visualization application. The fourth chapter describes an example visualizations. The following fifth chapter contains conclusion and future remarks. The user documentation and programmer documentation are located in sixth and seventh chapter.

2. Analysis

This chapter contains a short introduction into vector representation of documents and document clustering. Based on this representation it then describes the possibilities of algorithms for usage on single document and its parts.

2.1. Vector space and clustering concepts

Vector space model is a form of *ranked information retrieval*. The term *information retrieval* or information *search* in this context usually means finding the document(s) from a collection of documents, which satisfies needed information (based on definition in [1]). The result documents of the search are usually identified with respect to *search query*, a query that user formulated to express his information needs. In *ranked information retrieval* the *search algorithm* computes for each document the *rank*, how much does it satisfy the *search query*, and then – using this rank – it decides which documents are to be returned as a search result and in what particular order.

There exist more document models allowing ranked query that differs from each other by document representation and/or by the used search algorithm. In *Vector space model*, each document has assigned a representing *document vector*. This vector then represents the document content for the search algorithm. More detailed description of document vector could be found in section 2.1.2..

The *clustering* (or *data clustering*) is a process of grouping documents into *clusters*, where documents in one cluster should be as mutually similar as possible while documents taken from different clusters should be as different as possible from each other. In general, the clustering is not limited to documents, but can be used for various different types of data (images, music and other [2]). In this work we concern on clustering of document parts, such as chapters, paragraphs or sentences. More about it could be found in section 2.1.3..

2.1.1. Document vocabulary and terms

To create representing vector from a document, we first need to get information about its content. The common way of looking at document content, is considering it as a *bag of words*. This allows us to transform document text into *vocabulary* of *terms* and theirs frequencies (numbers of term occurrences inside individual documents). The first step is this process is a *tokenization* – a process of parsing the document text and splitting it into words (*tokens*). The *term* in this document represents a *token* in its *normalized* form. There are many normalization steps, which could use more or less complex computer linguistic approaches like *lemmatization*, *disambiguation*, *named entity identification* etc [1]. The steps mentioned below do not require such sophisticated approaches while still provide sufficient processing for the purpose of this work. The *normalization* usually uses several steps:

- *Diacritics* – Many languages (including Czech and Slovak ones) are using diacritics in their official form. However, in many cases the text is not written with correct diacritics or is written without any diacritics at all. This is mostly common on the internet, where many users don't use diacritics because of laziness, software limitations or habits. To solve this problems, the text could be stripped of diacritics and converted to 7-bit ASCII characters. The disadvantage of this approach is merging possibly different words into one common term.
- *Case-folding* – The words are converted to lower case. This allows to handle word with different case as the same. As a drawback, some abbreviations as "IT" could be converted to common terms.
- *Stemming* – Many different word occurrences in text could represent the same term, but differs in inflexion – prefixes or suffixes (for example *car* vs. *cars*). *Stemming* is a heuristic process that is an alternative to more complex *lemmatization*. It removes prefixes and suffixes from word and leaves only its base, called *stem*. For example, words *transporting*, *transported* and *transports* should be all considered as forms of common term *transport*.
- *Stop words filtering* – *Stop words* are usually referred as words, which occurs in language very often and usually don't have any essential meaning of their own. In English the typical words, that could be considered as stop words are *a*, *the*, *an*, *can* or *have* etc. By removing stop words, we could significantly decrease the space dimensionality – the number of different terms in index – and thus reduce process time and space required for indexes (according to *Rule of 30*[1]. During the text processing the *stop words* are usually predefined in *stop list*. The opposite approach could use whitelist of allowed terms in index instead of blacklist (stop list).

2.1.2. Document vector and indexing

In *Vector space model* the document d is represented by *n*-dimensional vector $\vec{v}_d$, where each dimension represents *weight* w_{d,t_i} of *term* t_i. In other words, the document vector is defined:

$$\vec{v}_d = \langle w_{d,t_1}, w_{d,t_2}, \ldots, w_{d,t_n} \rangle$$

The simplest way of computing term weights w_{d,t_i} is to assign number of

1 Rule of 30 states that the 30 most common words account for 30% of all terms in written text (Chapter 5 of [1]).

occurrences of term t_i in document. This is value called *term frequency* tf_{d,t_i}.

The *term frequency* is not optimal, because it considers all terms equally important. On the other hand, when we take documents about *insurance companies*, the term *insurance* would be probably very often in all of them, so it has almost none discriminating power. To reflect this, the *tf-idf weighting* could be used instead, which – according to [3] – produces better results in *data clustering*.

The *tf-idf weight* is defined as follows:

- *Document frequency* df_{t_i} – is defined as the number of documents containing the term t_i.
- *Inverse document frequency* idf_{t_i} – for term t_i is defined as:

  $idf_{t_i} = \log \frac{N}{df_{t_i}}$, where *N* is total number of documents.

The term weight w_{d,t_i} now can be defined using *tf-idf weight* as:

$$w_{d,t_i} = \text{tf-idf}_{d,t_i} = tf_{d,t_i} \cdot idf_{t_i}$$

Looking at the documents as vectors allows us to use standard vector operators and calculate lengths, or distances. Furthermore we can easily compute *similarity* of two documents, where two documents are *similar* when their document vectors are directionally close to each other. The commonly used measure for this purpose is the *cosine similarity*, which if for two documents d_1 and d_2 in [1] defined as

$$\text{sim}(d_1, d_2) = \frac{\vec{v}_{d_1} \cdot \vec{v}_{d_2}}{|\vec{v}_{d_1}| \cdot |\vec{v}_{d_2}|}$$

This *cosine similarity* allows us to compare relative distribution of terms in document independently of the document length.

2.1.3. Document clustering

Using *clustering* documents could be divided into *clusters*, where document pairs taken from the same *cluster* should be as similar as possible and documents in one *cluster* should be as dissimilar as possible from those in any other *cluster*.

Based on the organization of clusters, we can distinct two basic types of clustering:

- *Flat clustering* – the set of clusters is defined without any explicit organization between individual clusters.
- *Hierarchical clustering* – created clusters are organized in hierarchical structure.

Another important distinction of clustering is based on document assignment. From this point of views we can talk about:

- *Hard clustering* – where each document is assigned to exactly one cluster.
- *Soft clustering* – where document could be assigned to more clusters. In case the assignment is weighted, we are talking about so-called *Fuzzy clustering*.

The clustering of document collections is commonly used in web search, to speed up finding documents or pages matching the user's query. For example, when the user enters query *"apple"*, he or she may want to find information about the fruit, about the music recording company or about the computer manufacturer. But the search engine doesn't know which one the user wants, so by showing documents from different clusters in the first page, it is probable that the user will find some relevant document sooner.

Another example use of clustering is creating a summary of document collection. This is done by grouping similar documents into clusters and then replacing all documents in each cluster with surrogate piece of text, which represents them.

2.1.4. Spherical K-means clustering algorithm

K-means or its modification for information retrieval – *spherical k-means* – is one of the most important flat clustering algorithms, with hard assignment of documents. The algorithm starts by creating *k* random clusters, also known as *seeds.* Then it tries to optimize document assignment to clusters based on similarity of documents vector to their cluster *centroid*, computed as mean vector of assigned documents. The value of *k* is needed as a parameter. The algorithm could be formally describe as it is shown belows in Algorithm 1:

```
K-means (k, { $\vec{v}_{d_1}, \dots, \vec{v}_{d_n}$ })
begin
    create k initial random seed clusters
    recompute clusters centroids
    while not clusters are stable
    begin
        for each document
        begin
            reassign document to cluster with closest centroid
        end
        recompute clusters centroids
    end
    return k clusters
end
```

Algorithm 1: K-means

With each iteration in the while cycle, the algorithm reassigns document to the closest centroid and then recomputes the centroids. This way the centroids move around vector space to find their optimal assignments.

The *k-means* algorithm provides following advantages:

- It is one of the most used clustering algorithm, because it is relatively quick on large data sets. Its complexity is linearly proportional to the number of documents.
- Works well on numerical data.

On the other hand, it has several disadvantages:

- Result clusters have convex shapes only.
- The value of parameter *k* needs to be defined in advance.
- It provides only local optimization and could find assignment, which is suboptimal in global scale.
- Its performance depends on initial random selection of *k* clusters.
- Performs poorly on high-dimensional data (such as textual documents, [4]).

2.1.5. Hierarchical agglomerative clustering algorithm

Hierarchical agglomerative clustering (HAC) represents one of the basic hierarchical clustering algorithms. The algorithm starts with separate cluster for each document and in each step it creates new cluster, linking two most similar clusters (Algorithm 2).

The algorithm usually stops when all clusters have been linked and only one – forming the root of the cluster tree – is left. The algorithm could be enhanced with a stop condition, to stop when the similarity of clusters reaches some threshold, but in general it is hard to tell the proper value of the threshold.

Based on the way the similarity between two clusters is computed, HAC creates a whole class of algorithms. For example:

- *single link* – the similarity between two clusters is computed as the similarity of their most similar members
- *complete link* – the similarity between two clusters is computed as the similarity of their most dissimilar members
- *centroid* – the similarity between two clusters is computed from their centroid vectors

The HAC algorithm provides hierarchy of clusters representing the topic hierarchy of documents and it is considered as one to provide the best clustering results. However, it is not very usable on large data sets, because it is very slow.

```
HAC ({ $\vec{v}_{d_1}, \dots, \vec{v}_{d_n}$ })
begin
    create separate cluster for each document
    while count(clusters) > 1
    begin
        find two most similar clusters
        replace them with one new cluster, which is linked to them
    end
    return clusters
end
```

Algorithm 2: Hierarchical agglomerative clustering

As an opposition to HAC there is also *hierarchical divisive clustering*, which starts with only one cluster and iteratively splits largest clusters until individual documents or small enough clusters are reached.

2.2. Application of the theory to a single document

A single document could be viewed as a hierarchy of text fragments – chapters, paragraphs and sentences. Depending on a level of granularity, it could be also viewed as an ordered collection of chapters, ordered collection of paragraphs or as an ordered collection of sentences. This way it is possible to use any of algorithms originally invented for work on documents on these collections with only minor changes. The key differences are:

- Instead of single collection of separate documents, three parallel collections of chapters, paragraphs and sentences are used.
- Chapters, paragraphs and sentences are ordered and text fragments in them form a hierarchy. This structure is defined by the document itself. We could consider this hierarchy as a special type of hierarchical clustering where the similarity is derived from proximity.
- The vector in vector space model would represent not only whole document as in case of standard approach, but could also represent every single chapter, every single paragraph or even every single sentence, depending on currently working level of granularity.

The modified view on the document is shown in Figure 1. For the simplification, only one chapter level is considered. Possible sub-chapters are ignored and their text is taken as a content of top most chapter. For each node *d* of the hierarchy we can define:

- *Predecessor* of *d* – the closest previous text part on the same level of hierarchy.

- *Successor* of *d* – the closest next text part on the same level of hierarchy.
- *Parent* of *d* – the text part one level of hierarchy higher, which contains the node *d*.
- *Children* of *d* – ordered collection of text parts one level lower, which are contained by the node *d*.

Each node has assigned its own vector of term frequencies and representing vector of heights. The node representing a document part on different levels in the hierarchy would be referred as document, chapter, paragraph or sentence and the corresponding representing vector would be referred as *document*, *chapter*, *paragraph* or *sentence vector*.

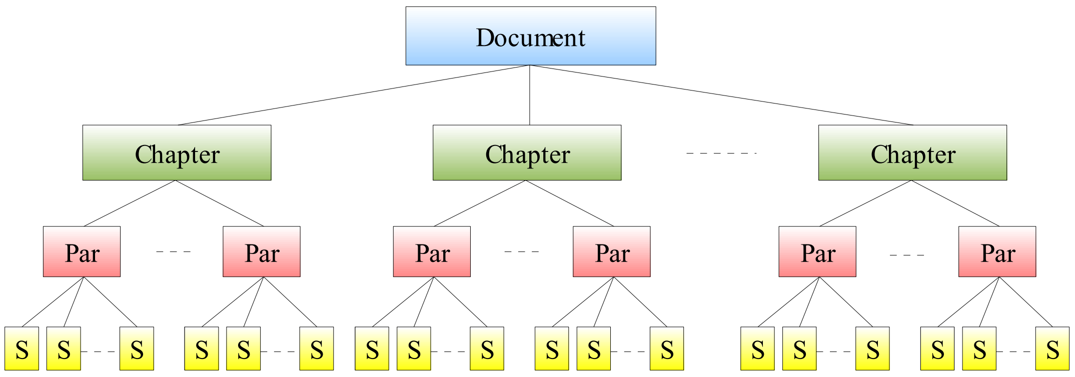


Figure 1: Overview of document hierarchy

Because of each inner node *d* of the hierarchy consists of a concatenation of its children, it holds that the document vector would be the vector sum of vectors of its paragraph children etc up to the paragraph vector would be the vector sum of vectors of its sentence children.

Using clustering on document chapters, paragraphs or sentences, we will get parallel hierarchies of clusters, organized by the real similarity of their content.

2.2.1. Term, vocabulary and indexing of single document

Because we are looking at the single document as a hierarchy, we need to slightly adjust the theory used in standard approaches, described in section 2.1..

All vectors of all granularity levels must be compatible – have the same dimension, because they will be used together in computations. Therefore, we would need a global vocabulary for all parts. On the other hand, each document part – chapter, paragraph or a sentence – would need to know its term occurrences to compute its vector.

The vector of document part could be define similar as document vector $\vec{v}_d$, with a *d* representing document part – chapter, paragraph or sentence. The difference would be in computing term weights using *tf-idf weight*. The *idf* would be derived from the number of paragraphs within the document and number of paragraphs containing given term. This *idf* computation would be used globally for all parts on all levels of hierarchy to get compatible numbers.

2.2.2. Example usage of vector model on single document

The document is converted to hierarchy of vectors that represents its content, and enables further processing and content analysis, such as:

- Computing similarity between chapters, paragraphs or sentences, discovering similar or the same parts that are spread across the document. This could be used for better orientation in documents, when the user would be provided with information about similar document parts, located in other chapters. For example, the user could get links to most similar chapters or link(s) to the closest chapter(s), with similarity bigger than defined threshold(s).
- Analyzing consistency of document flow. For example, if consecutive paragraphs in one chapter or consecutive chapters in the document are most similar to each other or not.
- Finding parts, which represents their parents the best. For example, finding a paragraph, which represents best the content of its chapter.

2.2.3. Example usage of clustering on single document

The use of clustering algorithms would group together similar content of the document. This would allow to:

- Identify the topics of the document and the parts which talks about them.
- Create summary of the document, by extracting topics from the document content. The topics could be represented by clusters, where each cluster would represent one topic. Summary would be created by taking part of document (few sentences) for each cluster.
- Extract keywords of the document in similar way, how summary could be created.
- Provide parallel navigation in the document by categorizing document parts into groups similar by content.
- Analyze consistency of document parts. For example, how much computed sets of clusters correspond to their positions in the document.

2.3. Generating document summary using clustering

As mentioned in previous section, clustering could be used to create document summary. The method described here is based on [5] with a difference in computing local and global similarities.

The summary of documents would be generated by selecting sentences from the text of the document. For each cluster, the sentence with maximal score is selected. The score of each sentence is computed as a weighted sum of following factors:

- Local similarity – It is computed as similarity between sentence and centroid of containing cluster. The more similar sentence to centroid, it is probable that it could best reflect the contents of the cluster.
- Global similarity – The similarity between sentence and the document, which ensures that the global context is reflected in selection.
- Sentence length – The length of summary could often be limited. This factors adds penalization for too short or too long sentences. The sentence length factor is defined as follows:

$$factor_{length} = \frac{1}{e^{\left|length(sentence) - length_{required}\right|}}$$

3. Implementation

This chapter describes implementation of simulation application SimDIS. It contains application design, describes used data structures and necessary interfaces.

3.1. Design and main decisions

The application should serve as a visualization and experimental tools with presumed rich interaction with the user. Therefore, the application with graphical user interface (GUI) is preferable to the console application.

Expected typical use case example usage of the application would be:

1. Opening of desired document.
2. Splitting document to its parts and computation of representing vectors.
3. Analyzing mutual similarities of objects at given level of granularity.
4. Computation of clusters of objects using selected clustering algorithm.
5. In many cases the user would probably like to visualize the data in some understandable way.
6. Explore the results and/or export them for further processing outside the application.

Because the user will probably work with one document more times in different scenarios, the application should import it locally. So the user would not have to search for the document in file system over and over.

Because the user may want to work on more documents, the application should allow to store more documents at the same time. To allow better management or moving between computers, the application should save documents grouped within projects, which are closely described later in 3.4.4..

To allow easy importation of documents, the application would support documents in HTML [6] file format. It is easy to analyze it and many third-party tools could be used to convert documents in almost any other format to it. It should be also possible to add support of more file formats later. Internally the document should be stored in custom format, best suiting the application needs for fast loading and processing. Import of document in any external format then should provide necessary conversion. The document list with document meta-data would be stored separately from the documents themselves for better performance. Used data structures are described in section 3.3..

As complex technical manuals could contain thousands and more documents parts, the application should support caching of already computed data to avoid repeating of time-consuming computations, like clusters and/or similarity matrices evaluation etc. Caches are closely described within project in section 3.4.4..

The application should be also easy to expand implemented features and to add new ones, because there exist more than one ways of text processing and visualization and it would be impossible to implement them all at once. The expandability of the application is discussed closely later in section 3.5..

3.2. Development tools

SimDIS is a GUI application written in C# language for Microsoft .Net Framework [7] with minimal required version 2.0. The application is targeted to run under Microsoft Windows [8] operating system. This platform was chosen mainly because it is most widely.

Some of the data are stored in SQLite database [9] using System.Data.SQLite library [10]. This database is small, doesn't require user to install any additional software and data can be stored within a single file.

The application was developed under Microsoft Visual Studio development environment [11] and for better source code management, Subversion [12] was used.

3.3. Data structures

The application uses several data structures for storing parsed document tree, vectors, terms and clusters. Almost all of them needs to be accessible from plugins to be able to normalize terms, to run data visualizations or to create clusters. This is the reason, why they are stored in *SimDIS.PluginInterface*.

The most notable data structures provided are:

- *Document* – the root of document hierarchy, consisting of *Chapters*, *Paragraphs* and *Sentences*. All of them are derived from *ADocumentPart*, an abstract class for document part.
- *DocumentVector* – vector representation of document part.
- *DocumentTerms* – the hierarchy parallel to Document, containing a set of *Terms* for each document part.
- *Term* – containing information about normalized term, its frequency and also list of original words used in document .
- *Cluster* – the hierarchy of clusters, created by clustering algorithm. Clusters contain the set of sub clusters in case of an internal node and a set of document parts in case of the leaves.

Detailed documentation of data classes with descriptions of methods and properties could be found in Code documentation on attached CD.

3.4. Application architecture

The Figure 2 bellow illustrates the architecture of the application from the logical view. Parts of the application are described in following sections.

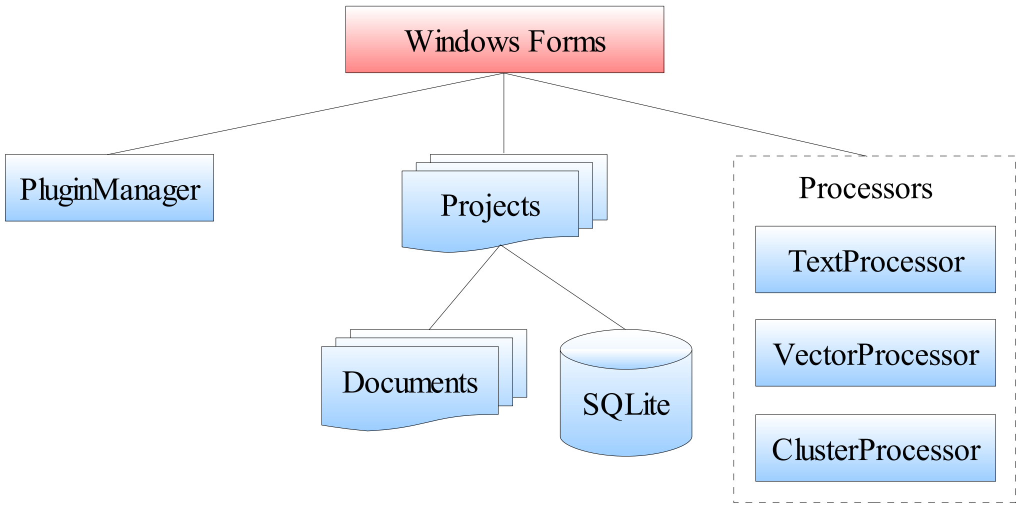


Figure 2: Overview of SimDIS core application

3.4.1. Windows forms

In the center of the application are *Windows Forms*, the GUI windows, which are responsible for interaction with the user. The most notable forms are:

- *MainWindow* – Displays list of documents stored in current project and allows document import and export. The list of documents is loaded from SQLite database and displayed using standard *DataGridView* component of .Net Framework.

- *DocumentWindow* – Represents one opened document and allows to further work it. The document is displayed as a HTML using *WebBrowser* component. The window also shows a list of visualizers, loaded from plugins, as a tool buttons to run visualization.

There are several other forms used in programs, mostly using standard components of .Net Framework.

Visualization plugins also contains window forms to visualize data to the user. They use their own forms, which are not limited or predefined by core application. This provides unlimited potential for data visualization.

3.4.2. Plugin manager

PluginManager is a static class, which takes care of all types of plugins. It loads all available plugins at the application startup, holds lists of them and also provides some useful methods to work with them. Plugins are more described in section 3.5..

3.4.3. Processors

In this application the processors are static classes in *SimDIS.Processors* namespace, which encapsulate text processing and creation of terms and document vectors. There are following processors:

- *TextProcessor* – is used for parsing document text and creating hierarchy of terms for document parts. The main methods is *createDocumentTerms(Document)*, which takes the document as parameter and walks through its parts and extracts words from them. From the words, the normalized terms are created, using active normalization plugins. If the normalization would result in and empty string, the word is considered as a stop word. At the end, the *DocumentTerms* object is created.

- *VectorProcessor* – is used for creating document vector hierarchy from already prepared hierarchy of document terms – the *DocumentTerms* object. The *createDocumentVectors(Document, DocumentTerms)* walks through document parts and from prepared terms it creates and assigns instances of *DocumentVector* to them.

- *ClusterProcessor* – is used for creating labels and titles of clusters. The clusters themselves are created using clustering algorithm plugins. The labels are chosen from terms in centroid vector of the cluster using their frequencies. The titles are selected as a sentences most similar to centroid of each cluster.

3.4.4. Projects

The projects are used as a container for storing documents and user work. They also allow to easy transfer the work from one computer to another and to have multiple parallel projects saved in application at the same time.

Physically each project is saved as a separate sub-directory of *projects* directory located under the application directory. It contains:

- XML file project.xml with basic description of the project (for example, name of the project). It is used only to quickly identify the project. The example of the file is presented in Figure 3.

```
<?xml version="1.0" encoding="utf-8"?>
<project>
      <project-name value="The name of project" />
</project>
```

Figure 3: Example of project.xml file

- SQLite database in file storage.sq3, where for example list of documents is stored. SQL provides better and faster access to data collections or changing data than XML.

- Saved documents and computation cache as separate files, using serialization of .Net Framework. The serialization has been used because it is much faster than SQLite, when storing lots of textual data and object hierarchy. The name of each cache file is derived from the document to which it belongs, cache name and ".cache" extension. For example: *document_3_SimilarityParagraphs.cache*

The list of projects is accessed through static class *SimDIS.Project. ProjectManager*, which could check the *projects* directory and get the list of all projects by calling *getProjectList()*.

Once opened, the project is represented by *Project* class in *SimDIS.Project* namespace and is partially defined in more files to increase readability of the code. The *Project* class provides functionality regarding the project and its documents, such as:

- Loading and saving the project itself – this is done automatically in its constructors, where from the parameters it knows whether is should load from directory or a new project is being created.
- Managing SQLite database – the database is accessible by the *DocumentStorage* property, which returns *DbConnection*. By accessing the property, database connection is automatically initialized. When it is accessed for the first time and the database file doesn't exist, the file is automatically created including SQL schema. This is done by *initializeDb()* and *createDbSchema()* methods.
- Saving document to project using *saveDocument()*, loading using *getDocument()* and deleting using *deleteDocument()*. Title and name of the document is saved into database and the document it selves is stored into separate file using serialization of .Net Framework.

Getting the list of authors and titles of stored documents using *getDocumentTitles()*. The list is loaded from database.

3.5. Plugin interfaces

To meet the requirements for a tool for testing, visualizing and analyzing impact of vector model and clustering used on document and its parts – The SimDIS application implements several tools for loading documents, parsing them, indexing their sub-parts, analyzing their mutual similarities, clustering them and visualizing the results. For better extensibility and easier maintenance in the future the

application uses plugins for individual visualization tools, implemented clustering algorithms etc. All plugins are loaded at the start of the application from external dynamic-link libraries (DLL's). This allows adding new features without the need for source code of core application and its recompilation.
The features which may be desirable to be added or modified later are:

- More supported file formats – the file types for importing and exporting documents to/from the application. For example, it would be suitable to allow processing documents in some of e-book format as epub or another XML based format.

- Term normalization – the application should allow to add more complex filtering steps within the term normalization process (2.1.1.).

- Clustering algorithms – there are many various algorithms, which vary in way the clusters are computed.

- Clustering and non-clustering visualizations – would allow to add different ways to look at document and its content.

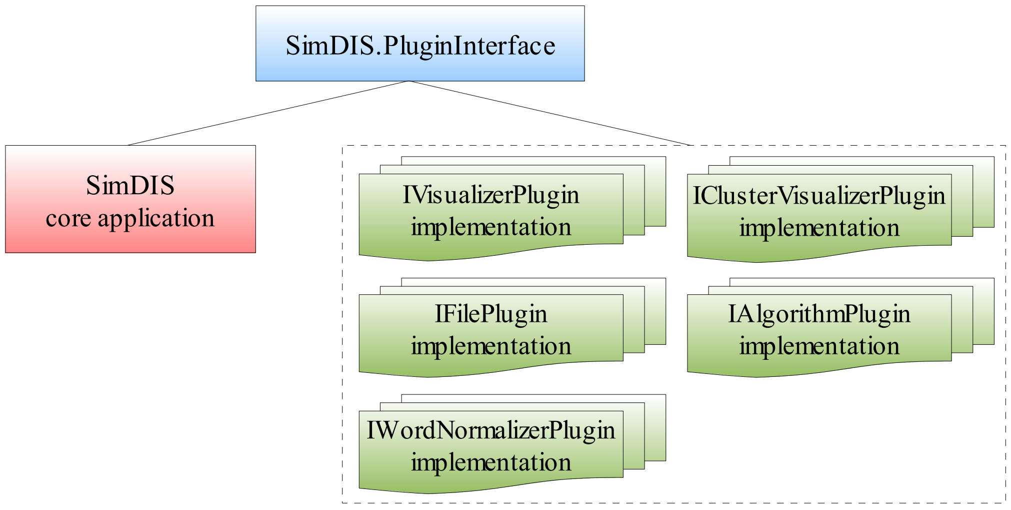

SimDIS.PluginInterface
SimDIS core application
IVisualizerPlugin implementation
IClusterVisualizerPlugin implementation
IFilePlugin implementation
IAlgorithmPlugin implementation
IWordNormalizerPlugin implementation

Figure 4: Basic concept of plugin implemetation

Figure 4 shows the concept of plugin implementation, where the left box represents the application core part, which links the *SimDIS.PluginInterface* library depicted by box at the top of the picture, which makes accessible all necessary interfaces plugins trough their respective and maintains data, needed by plugins to work with document. Boxes at the right side enclosed in dashed box represent the implementations of interfaces via plugins in application context.

The plugin could be one of following type:

- *IFilePlugin* – implements support for new file types to importing and exporting documents.

- *IWordNormalizerPlugin* – implements one step in term normalization process, like case-folding, stemming or stop words filtering, etc. Normalization steps are closely described in section 2.1.1..
- *IAlgorithmPlugin* – the implementation of clustering algorithms.
- *IVisualizerPlugin* – the non-cluster visualization, visualizing mainly chapter vectors, paragraph vectors, sentence vectors or document terms. To this category fits almost anything else, for example, plugin that displays similarity matrix of paragraphs or plugin that shows table of processed document terms.
- *IClusterVisualizerPlugin* – for the visualization of clusters. For example, plugin that displays clustered document parts as a browseable tree.

Each plugin is represented by one or more classes, where each one implements one or more of previous interfaces. Class(es) implementation is packed in DLL, which must be located in *plugins* directory and could contain more than one plugin class.

The plugin directory is scanned at the startup of application and found plugins are loaded and registered automatically. This is done by *SimDIS.Plugins.PluginManager* using *System.Reflection.Assembly* and *System.Activator* of .Net framework. In loading process, each class from each DLL is checked, whether it implements some of the plugin interfaces mentioned above and if so, it is registered as plugin – added to appropriate list within the PluginManager. Later in application those lists are used to generate menu items, get supported file type or normalize words. This lists are accessible as following properties:

- *Files* – list of file active *IFilePlugin* plugins.
- *WordNormalizers* – list of active *IWordNormalizerPlugin* plugins.
- *Algorithms* – list of active *IAlgorithmPlugin* plugins.
- *DocumentVisualizers* – list of active *IVisualizerPlugin* plugins.
- *ClusterVisualizers* – list of active *IClusterVisualizerPlugin* plugins.

The *PluginManager* also provides methods to process some of the plugin functionality:

- *getNormalizedWord()* – takes word, applies all active normalization plugins to it and returns the result
- *openDocument()* – reads document from file, using appropriate file plugin

- *saveDocument()* – saves document to an external file, using appropriate file plugin

To allow DLL to contain more than just plugin classes and for better orientation in files and classes, a naming convention is enforced:

- The name of plugin DLL must end with “Plugin” (for example, BasicFilePlugin.dll) to be loaded by application.
- The name of plugin class must end with one of: “Visualizer”, “VisualizerCluster”, “File”, “Algorithm” or “Normalizer”. Otherwise the class will not be loaded.

3.6. Implemented plugins

As a part of this application, various plugins were implemented as a normalizers, visualizers and clustering algorithms. This section describes some of them.

3.6.1. Term normalizers

The normalization is used for grouping together words with similar meaning, but different written form. It is part of document processing, which is described in section 2.1.1..

Term normalizers are implementations of *IWordNormalizerPlugin* with *normalize(string)* being the main method. The method takes word as a parameter and applies one step of normalization on it.

The execution of normalizers is piped in order concluded by default from their *getPriority()* value. The explicit order could be forced by the user by setting of SimDIS.Plugin.NormalizersOrder property, which is closely described in user documentation in section 6.3..

Following normalizers were implemented and are invoked in following order:

- *ToAsciiNormalizer* – removes diacritics.
- *ToLowerNormalizer* – converts all characters to lower-case.
- *TrivialStemmerNormalizer* – represents a trivial implementation of word stemming. It tries to remove English, Slovak and Czech prefixes and suffixes to get their stems. It uses regular expressions to find first suitable prefix and suffix to remove. Because it mixes multiple languages, it cannot guarantee that the stem would be generated always correctly. On the other hand, this cannot be guaranteed even by more complex stemmers.
- *StopWordsNormalizer* – compares word against stop list located in stoplist.txt file. If given word is a stop word, it results and empty string. The stop list

contains stop words from English, Slovak and Czech language in lowercase 7-bit ASCII form and their stems created by the *TrivialStemmerNormalizer*.

3.6.2. Clustering algorithms

Clustering algorithms are implementations of *IAlgorithmPlugin* and their main purpose it to create clustering from set of document parts (*ADocumentPart*) in method *computeCluster()*.

There are two basic groups of clustering algorithms implemented:

- *Flat clustering* – represented by *KMeansAlgorithm*, the implementation of K-Means clustering algorithm.
- *Hierarchical clustering* – representation by *CentroidHACAlgorithm*, the implementation of hierarchical agglomerative clustering using centroid vectors.

The result of clustering algorithm is a hierarchy of *Cluster* objects. In flat algorithms the hierarchy consists of one level under fictional cluster root.

3.6.3. Document visualizers

Document visualizers, the implementations of *IVisualizerPlugin*, are used for non-cluster visualization. Some of the visualizations, that were implemented:

- *WordsHtmlVisualizer* – allows to explore results of document processing, by showing tables of terms for each document part embedded in text. The output is displayed using combination of HTML and JavaScript in *WebBrowser* component.
- *DocumentTermsVisualizer* – shows the document term vocabulary table using *DataGridView*.
- *ChapterSimilarityVisualizer*, *ParagraphSimilarityVisualizer* – display a similarity matrix between chapters or paragraphs. The output is displayed in *SimilarityWindow* form using *DataGridView* component or alternatively as image, where each pixel represents one value of similarity matrix in gray scale. The pixel is the brighter the bigger the similarity is. While grid view provide exact information about similarities, the image viewer allows obtaining quick overall insight into similarity distribution.
- *ChapterConsistencyVisualizer*, *ParagraphConsistencyVisualizer* and *SentenceConsistencyVisualizer* – compute similarity of document part with its predecessor, successor and with its parent. The output is displayed in *GridWindow* form.

Many of above mentioned visualizers also provides ability to export data for further analysis. The formats of exported files could vary with visualized data. For example, the similarity matrix is possible to export as HTML, CSV and even PNG file.

3.6.4. Cluster visualizers

Cluster visualizers are implementations of *IClusterVisualizerPlugin*. Their purpose it to provide visualization of document parts clustering. Examples of implemented cluster visualization:

- *SentenceTreeVisualizerCluster* – displays clustering of sentences in browsable tree.
- *ParagraphTreeVisualizerCluster* – displays clustering of paragraphs in browsable tree.
- *ChapterTreeVisualizerCluster* – displays clustering of chapters in browsable tree.
- *ChapterSummaryVisualizerCluster*, *ParagraphSummaryVisualizerCluster* and *SentenceSummaryVisualizerCluster* – using the method described in section 2.3. this visualizer allow to generate text summary. The summary is generated from clusters of corresponding document parts – clusters of chapters, paragraphs of sentences.

Cluster visualizers allows to set the *threshold* for cutting the cluster hierarchy when using hierarchical clustering. For example, the threshold of 0.4 will cut the cluster hierarchy, where clusters with similarities of their siblings bigger than 0.4 will be left as whole and those with smaller will be split.

4. Usage example analysis

This section provides an example of application usage and compares some of the visualization results with respect to granularity level. The example is made on book The Underground City from Jules Verne, which consists of 19 chapters, 1043 paragraphs and 2330 sentences.

4.1. Non-clustering visualizations

After importing the document into application and generating document terms and vectors, using DocumentTermsVisualizer we could see, that document contained 44051 words, which resulted in 4010 terms in document term table. As could be seen in Table 1, which shows first 12 terms ordered by their word count, the first 11 are stop words and the first non stop term is on twelve position. The total number of stop word terms was 536 and they stopped 25526 word occurrences.

Id	Term	Word count	Stop word	Inverse frequency
22	the	3010	True	0,1183334
32	of	1544	True	0,2369451
20	to	1197	True	0,2681921
125	and	900	True	0,31455
30	a	746	True	0,3626023
61	was	629	True	0,4528527
105	in	628	True	0,4243079
158	that	480	True	0,4923612
150	it	423	True	0,559308
112	his	389	True	0,6330943
93	had	377	True	0,6439521
47	harry	354	False	0,5162734

Table 1: First 12 terms in document ordered by word count

With use of ChapterConsistencyVisualizer, ParagraphConsistencyVisualizer and ChapterConsistencyVisualizer we could get similarity of document part with its parents. The average values are presented in Table 2 and this values confirms that the document contains a lot of short paragraphs with 2.23 sentences in average, because the sentences are very similar to theirs paragraphs.

	Similarity with paragraph	**Similarity with chapter**	**Similarity with document**
Characters	x	x	0,5614398
Paragraphs	x	0,1919763	0,1199073
Sentences	0,6228683	0,1346669	0,0840155

Table 2: Similarity of document parts with theirs parents

4.2. Visualization with use of clustering

Clustering could be used on chapters, paragraphs or sentences, which gives us three parallel cluster structures of the document content. Therefore it may be interesting to compare them.

The application so far implements two clustering algorithms mentioned before in section 2.1. and Table 3 shows the time in seconds spent on their computation. It is clear that with deeper granularity the computing time raises rapidly and with Centroid HAC even more. The values are measured on common desktop PC with 1.66 GHz dual core processor.

Algorithm used	**On chapters**	**On paragraphs**	**On sentences**
K-means, k=10	0,9375 sec.	5,984375 sec.	13,82813 sec.
Centroid HAC	1,25 sec.	32,79688 sec.	131,375 sec.

Table 3: Time spent on computation of clusters in seconds

Clusters are used in automated generating of document summary, introduced in section 2.3., by cluster visualizers ChapterSummaryVisualizerCluster, ParagraphSummaryVisualizerCluster and SentenceTreeVisualizerCluster. In Table 4 we can see, that the algorithms spent both above the same time on generating summary from clusters over different granularity. The required sentence length was set to 70 characters.

Algorithm	Chapters	Paragraphs	Sentences
K-means, k=10	0,515625 sec.	0,59375 sec.	0,546875 sec.
Centroid HAC	0,84375 sec.	0,75 sec.	0,75 sec.

Table 4: Time spent on generating document summaries

The summary generated using K-means is enclosed in Appendix A: Generated summaries from K-means algorithm. The results are pretty solid and for paragraphs and sentences very similar, but for chapters the quality is noticeable lower.

The summary generated using Centroid HAC in comparison with previous from K-means on the other hand performs poorly. The results are enclosed in Appendix B: Generated summaries from centroid HAC algorithm.

5. Conclusion

5.1. Project contribution

This work took a closer look at a possibilities of applying vector-space model and clustering techniques on individual parts within a single document. This application of vector model and clustering is not an usual approach and there hasn't been much research in it. The analysis has shown, that a transformation from document collection to intra-document analysis needs only a minor adjustment and the theory and algorithms could be transformed.

Implemented application with already built-in visualizing tools provides and easy way for testing and analyzing the content of documents in interactive form. This could help to measure the suitability of used techniques. The data could be analyzed within the application itself or could be exported for further processing.

5.2. Visualization results

Using list of terms, it was possible to verify the document indexation and by the frequencies of the terms, to identify possible candidates for addition to stop list.

The image presentations of similarity matrices show that real documents contain often similar areas located far away each from other and so the easy navigation between them would be helpful. The same visualization could be used as a hint for document structuring to describe similar topics closer together.

Comparing K-mean and HAC algorithms, the HAC was able to detect parts which are dissimilar from the rest of the content and so it could better detect the topics of the document. However, when generating summary the K-mean was giving better results, because it was able to select more relevant sentences, which were better reflecting the overall content of document.

Generated summaries showed, that for longer texts the summaries from sentence level clusters and paragraph level clusters were very similar in quality. The number of paragraphs in lower than the number of sentences and therefore the clustering of paragraphs would take significantly less time.

5.3. Possible future work

The possible ways of visualization are practically unlimited. Therefore, it is probable that more of them would be added in future. It could be also suitable to implement more clustering algorithms.

The implemented plugins for word stemming provides only basic approach. To improve the token normalization, some lemmatizers dependent on document language could be involved into the process.

The supported HTML format proven itself as sufficient for testing purposes. There are a lot of tools, which allows to convert other types of documents to it. On the other hand, there are a lot of free electronic books available targeted for electronic book readers. So it could be handy to add native support for some of them.

The possible improvements mentioned so far are mainly related to plugins. However there are possible improvements also within the core application. For example, it may be interesting to add support for different similarity measures and their comparison.

6. User documentation

6.1. System requirements

This application designed to run under Microsoft Windows XP SP3, Windows Vista SP1 and Windows 7 in 32bit version.

To run the application, Microsoft .Net Framework 2.0 or later is required to be installed.

6.2. Installation

The application needs to be installed on writable drive in a folder, where the application would be allowed to read and write files. The installation could be done via provided setup.exe wizard or simply by extracting SimDIS.zip package. Both files could be found on attached CD.

No other steps are required to run the application.

6.3. Properties

Some of application parameters could be configured in application.properties file in application directory. In this file, they will be used globally for the whole application. The properties could be overloaded in project.properties files, located in directory of each project.

The parameter is defined in form *PARAMETER_NAME=VALUE*. The character # at the beginning of line is taken as a row comment and so the row is skipped.

Available parameters and their values could be found in Table 5. The parameter names are case-sensitive, but there could be white spaces before and after it.

Parameter name	**Description**	**Values**
SimDIS.Plugin.NormalizersOrder	Defines which plugins should be used for normalization and their order. If empty, all are used in default order. One plugin could occur multiple times.	Plugin1|Plugin2|Plugin3
KMeansAlgorithm.K	The K value of K-Means algorithm – the number of clusters to create. If 0 or empty, it tries to determine the number automatically.	integer
SimilarityWindow.MaxZoomLevel	Maximal level for zooming the image in similarity window. Default value is 10.	integer
StopWordsNormalizer.StopListFile	Allows to define custom file name of stop list. By default stoplist.txt is used.	string

Table 5: Configurable parameters in properties

6.4. Working with the application

The application is started by executing SimDIS.exe file.

6.4.1. Starting a project

At the application startup, the welcome screen is shown to user (Figure 5). On this screen, the user can create and start a new project or to load (Figure 6) data from existing projects. The projects are represented as a directories in *projects* directory.

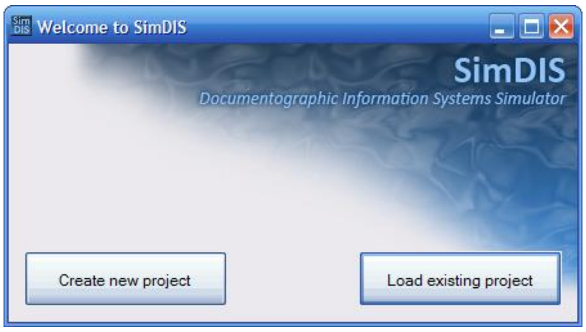
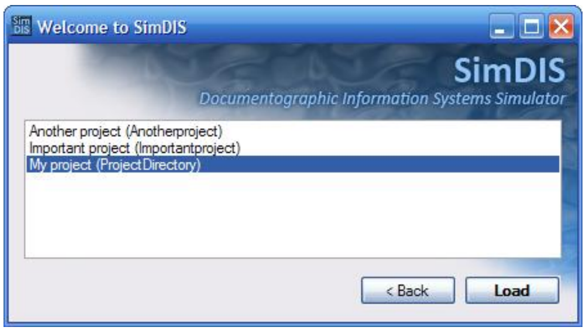


Figure 5: Welcome screen



Another project (Anotherproject)
Important project (Importantproject)
My project (ProjectDirectory)

Figure 6: Project loading selection

When creating the project, the name of directory is generated automatically from the project name. In could be later renamed or copied using common tools for browsing file system in Windows.

6.4.2. Document list

After the project is loaded, the main window with list of documents is shown (Figure 7). The list contains author and title of documents and also number of chapters, paragraphs and sentences.

Here it is possible to manage documents, stored within the project:

- Open selected document – opens the first document that is selected in list.
- Import document – opens file dialog and imports one or more documents.
- Delete selected document – deletes all documents selected in list.

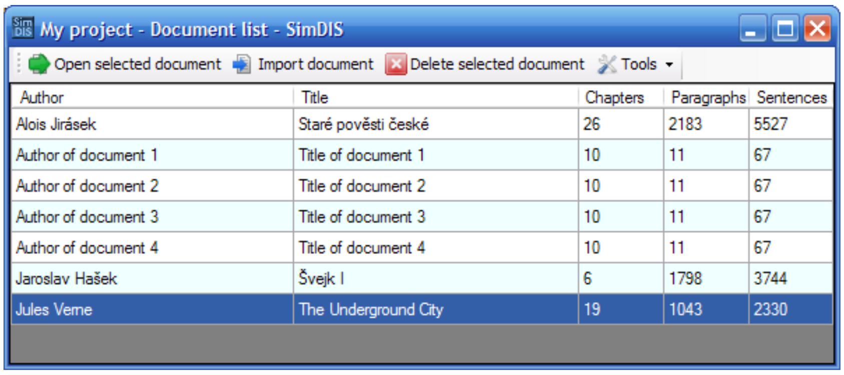
Under section Tools in menu, the user could find following:

- Clear project cache – deletes all cache files withing the project.

- Edit properties – opens a window with project properties for editation. This is changing only property file located within the project and not the global one. After save the properties are automatically reloaded.

- Test normalizers – opens a window, which allows to load file containing words – each on single line – and then by use of selected normalizer it saves the result to output file. For example, loading a stop words file, applying stemmer on it and saving the result into new stop words file.

- About SimDIS – information about the application.



Author	Title	Chapters	Paragraphs	Sentences
Alois Jirásek	Staré pověsti české	26	2183	5527
Author of document 1	Title of document 1	10	11	67
Author of document 2	Title of document 2	10	11	67
Author of document 3	Title of document 3	10	11	67
Author of document 4	Title of document 4	10	11	67
Jaroslav Hašek	Švejk I	6	1798	3744
Jules Verne	The Underground City	19	1043	2330

Figure 7: Document list window illustration

6.4.3. Document window

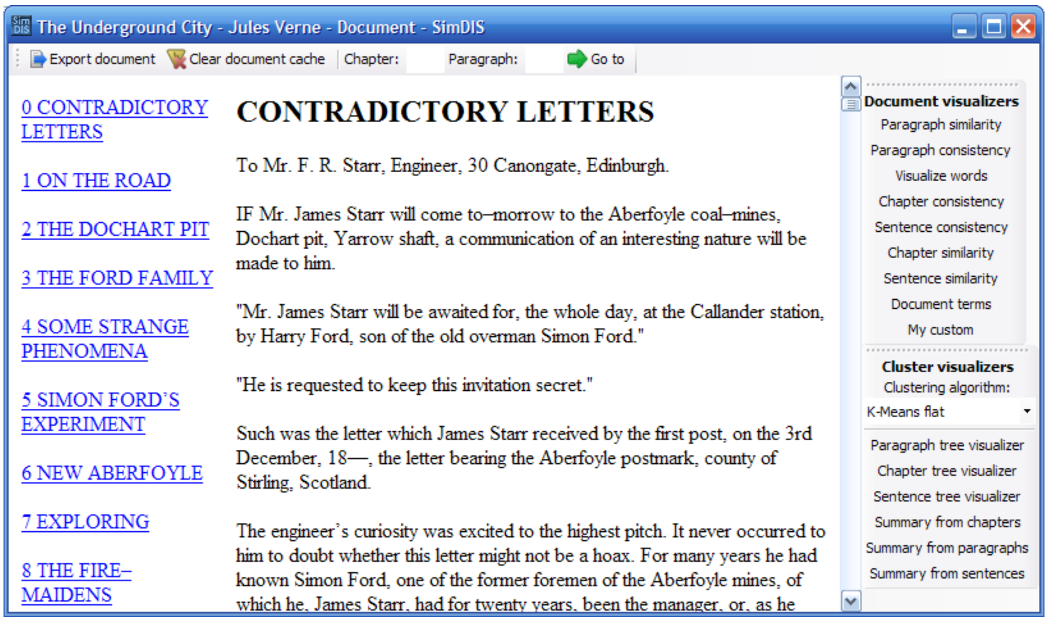
By opening document from document list, the Document window is shown (Figure 8). It shows full text of the document and allows to run visualization on it.

Document window offers following actions in top menu:

- Export document – opens save file dialog and then exports current document into selected place.

- Clear document cache – deletes all cache files, which belongs to current document.

- Go to – it is possible to enter the number of chapter and paragraph and by clicking on Go to button, the document text would be scrolled to this document part.


The Underground City - Jules Verne - Document - SimDIS

Export document | Clear document cache | Chapter: | Paragraph: | Go to

0 CONTRADICTORY LETTERS
1 ON THE ROAD
2 THE DOCHART PIT
3 THE FORD FAMILY
4 SOME STRANGE PHENOMENA
5 SIMON FORD'S EXPERIMENT
6 NEW ABERFOYLE
7 EXPLORING
8 THE FIRE–MAIDENS

CONTRADICTORY LETTERS

To Mr. F. R. Starr, Engineer, 30 Canongate, Edinburgh.

IF Mr. James Starr will come to–morrow to the Aberfoyle coal–mines, Dochart pit, Yarrow shaft, a communication of an interesting nature will be made to him.

"Mr. James Starr will be awaited for, the whole day, at the Callander station, by Harry Ford, son of the old overman Simon Ford."

"He is requested to keep this invitation secret."

Such was the letter which James Starr received by the first post, on the 3rd December, 18—, the letter bearing the Aberfoyle postmark, county of Stirling, Scotland.

The engineer's curiosity was excited to the highest pitch. It never occurred to him to doubt whether this letter might not be a hoax. For many years he had known Simon Ford, one of the former foremen of the Aberfoyle mines, of which he, James Starr, had for twenty years, been the manager, or, as he

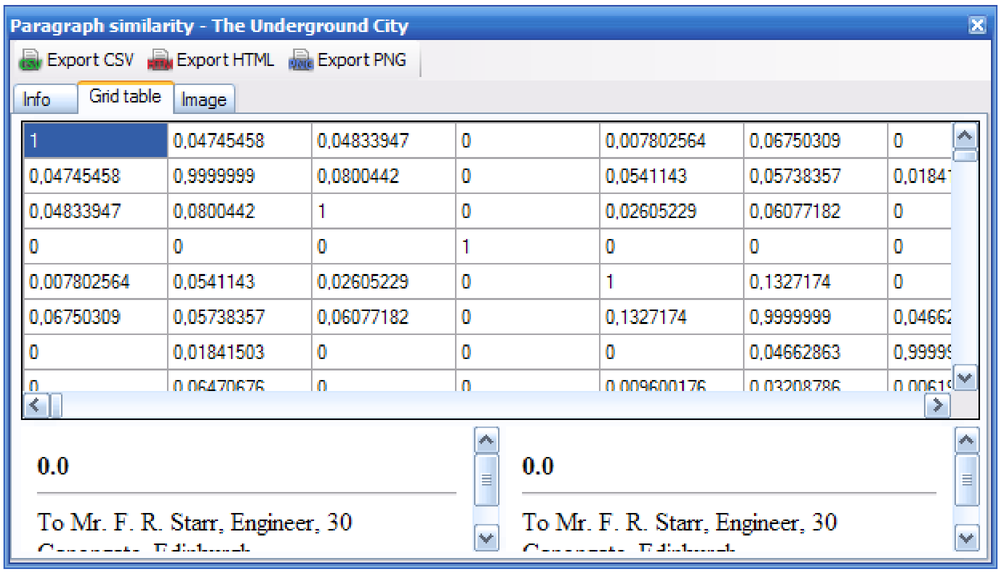
Figure 8: Document window illustration

The visualizers plugins are automatically registered and are offered to user in side menu, from which he can execute them. When visualization is executed, it usually opens its own specific window.

The visualizations of clusters are related for currently selected clustering algorithm.

6.4.4. Chapter, paragraph and sentence similarity

All three visualizers use the same window, which display the similarity as table, as in Figure 9, or as and image, as in Figure 10.


Paragraph similarity - The Underground City

1	0,04745458	0,04833947	0	0,007802564	0,06750309	0
0,04745458	0,9999999	0,0800442	0	0,0541143	0,05738357	0,0184
0,04833947	0,0800442	1	0	0,02605229	0,06077182	0
0	0	0	1	0	0	0
0,007802564	0,0541143	0,02605229	0	1	0,1327174	0
0,06750309	0,05738357	0,06077182	0	0,1327174	0,9999999	0,0466
0	0,01841503	0	0	0	0,04662863	0,9999
0	0,06470676	0	0	0,009600176	0,03208786	0,0061

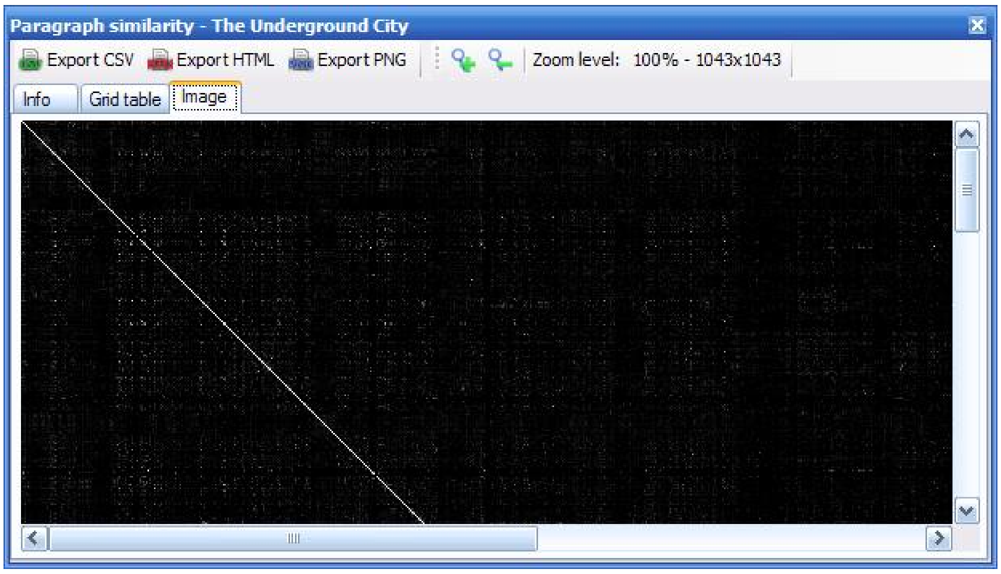
0.0

To Mr. F. R. Starr, Engineer, 30

0.0

To Mr. F. R. Starr, Engineer, 30

Figure 9: Similarity window showing grid view



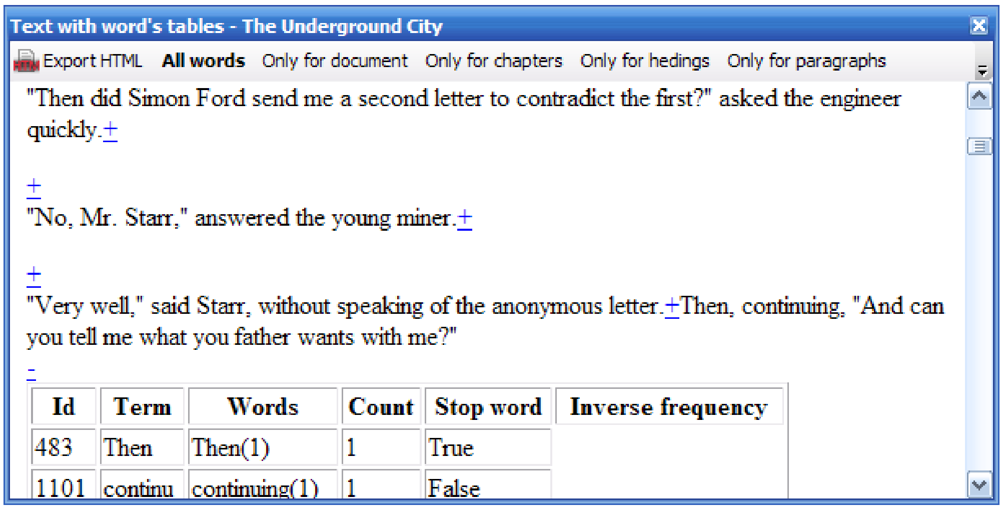
Paragraph similarity - The Underground City
Export CSV
Export HTML
Export PNG
Zoom level: 100% - 1043x1043
Info
Grid table
Image

Figure 10: Similarity window showing image

The user could play with it and event export it into CSV, HTML or PNG for further processing in external programs.

6.4.5. Words and terms visualization

There are two direct visualizations for showing term vocabulary. First – Visualize words – shows whole text in a HTML window (Figure 11) and adds hidden "+" to every document part. After clicking on it, the term table and vector of corresponding document part is shown. The "+" changes to "–" and when clicked, it hides back the table.



"Then did Simon Ford send me a second letter to contradict the first?" asked the engineer quickly.+

+
"No, Mr. Starr," answered the young miner.+

+
"Very well," said Starr, without speaking of the anonymous letter.+Then, continuing, "And can you tell me what you father wants with me?"
-

Id	Term	Words	Count	Stop word	Inverse frequency
483	Then	Then(1)	1	True	
1101	continu	continuing(1)	1	False	

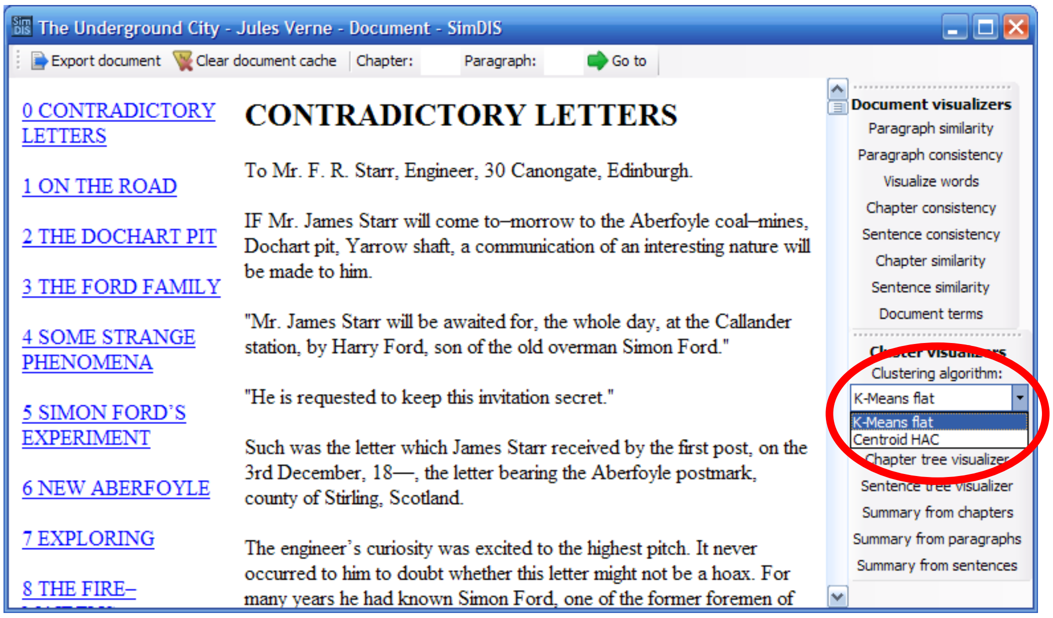
Figure 11: HTML word visualization window

By selecting in top menu, the user could affect to which parts the "+" are displayed.

The second visualization just shows global dictionary and allows to export it.

6.4.6. Working with clusters

The clustering algorithm, which is to be used for creating clusters could be selected in menu in document window as shown on Figure 12.



The Underground City - Jules Verne - Document - SimDIS

Export document | Clear document cache | Chapter: | Paragraph: | Go to

0 CONTRADICTORY LETTERS
1 ON THE ROAD
2 THE DOCHART PIT
3 THE FORD FAMILY
4 SOME STRANGE PHENOMENA
5 SIMON FORD'S EXPERIMENT
6 NEW ABERFOYLE
7 EXPLORING
8 THE FIRE–

CONTRADICTORY LETTERS

To Mr. F. R. Starr, Engineer, 30 Canongate, Edinburgh.

IF Mr. James Starr will come to–morrow to the Aberfoyle coal–mines, Dochart pit, Yarrow shaft, a communication of an interesting nature will be made to him.

"Mr. James Starr will be awaited for, the whole day, at the Callander station, by Harry Ford, son of the old overman Simon Ford."

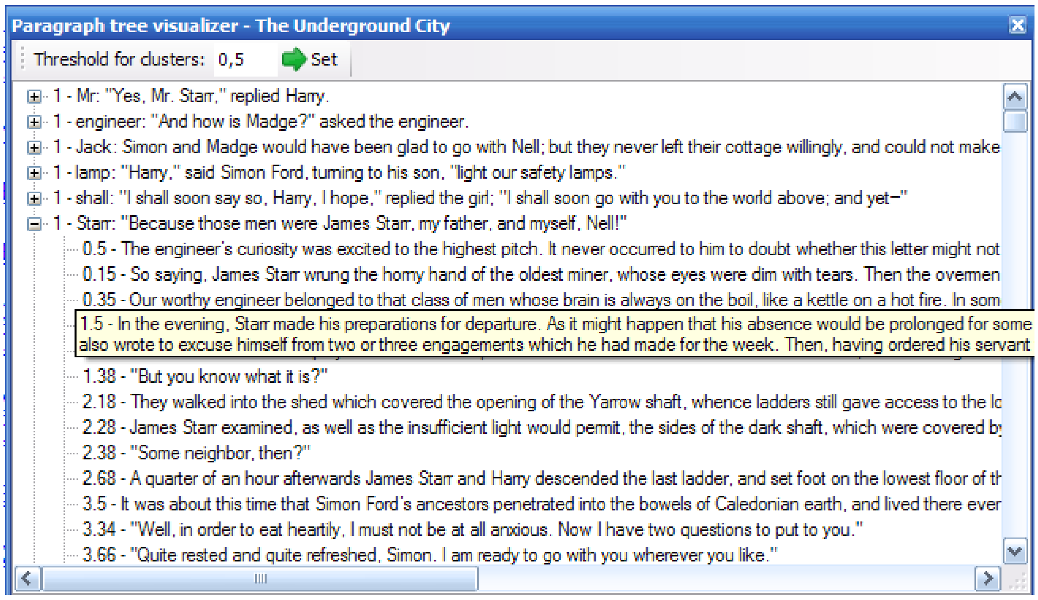
"He is requested to keep this invitation secret."

Such was the letter which James Starr received by the first post, on the 3rd December, 18—, the letter bearing the Aberfoyle postmark, county of Stirling, Scotland.

The engineer's curiosity was excited to the highest pitch. It never occurred to him to doubt whether this letter might not be a hoax. For many years he had known Simon Ford, one of the former foremen of

Figure 12: Selecting clustering algorithm

Then the cluster could be seen clicking on Chapter tree, Sentence tree or Summary tree visualizers. The cluster tree window would be opened (Figure 13).



Paragraph tree visualizer - The Underground City

Threshold for clusters: 0,5 Set

```
⊞ 1 - Mr: "Yes, Mr. Starr," replied Harry.
⊞ 1 - engineer: "And how is Madge?" asked the engineer.
⊞ 1 - Jack: Simon and Madge would have been glad to go with Nell; but they never left their cottage willingly, and could not make
⊞ 1 - lamp: "Harry," said Simon Ford, turning to his son, "light our safety lamps."
⊞ 1 - shall: "I shall soon say so, Harry, I hope," replied the girl; "I shall soon go with you to the world above; and yet–"
⊟ 1 - Starr: "Because those men were James Starr, my father, and myself, Nell!"
    0.5 - The engineer's curiosity was excited to the highest pitch. It never occurred to him to doubt whether this letter might not
    0.15 - So saying, James Starr wrung the horny hand of the oldest miner, whose eyes were dim with tears. Then the overmen
    0.35 - Our worthy engineer belonged to that class of men whose brain is always on the boil, like a kettle on a hot fire. In som
    1.5 - In the evening, Starr made his preparations for departure. As it might happen that his absence would be prolonged for some
    also wrote to excuse himself from two or three engagements which he had made for the week. Then, having ordered his servant
    1.38 - "But you know what it is?"
    2.18 - They walked into the shed which covered the opening of the Yarrow shaft, whence ladders still gave access to the lo
    2.28 - James Starr examined, as well as the insufficient light would permit, the sides of the dark shaft, which were covered by
    2.38 - "Some neighbor, then?"
    2.68 - A quarter of an hour afterwards James Starr and Harry descended the last ladder, and set foot on the lowest floor of th
    3.5 - It was about this time that Simon Ford's ancestors penetrated into the bowels of Caledonian earth, and lived there ever
    3.34 - "Well, in order to eat heartily, I must not be at all anxious. Now I have two questions to put to you."
    3.66 - "Quite rested and quite refreshed, Simon. I am ready to go with you wherever you like."
```

Figure 13: Cluster tree visualizer window

In this window, user could browse through the clusters. Each node represents one cluster, where first is the similarity of its sub-clusters or 1 in for leaf and then its label. The leaf cluster contains member document parts, where the first number represents their location in document.

By hovering on document part, a title with full text is shown.

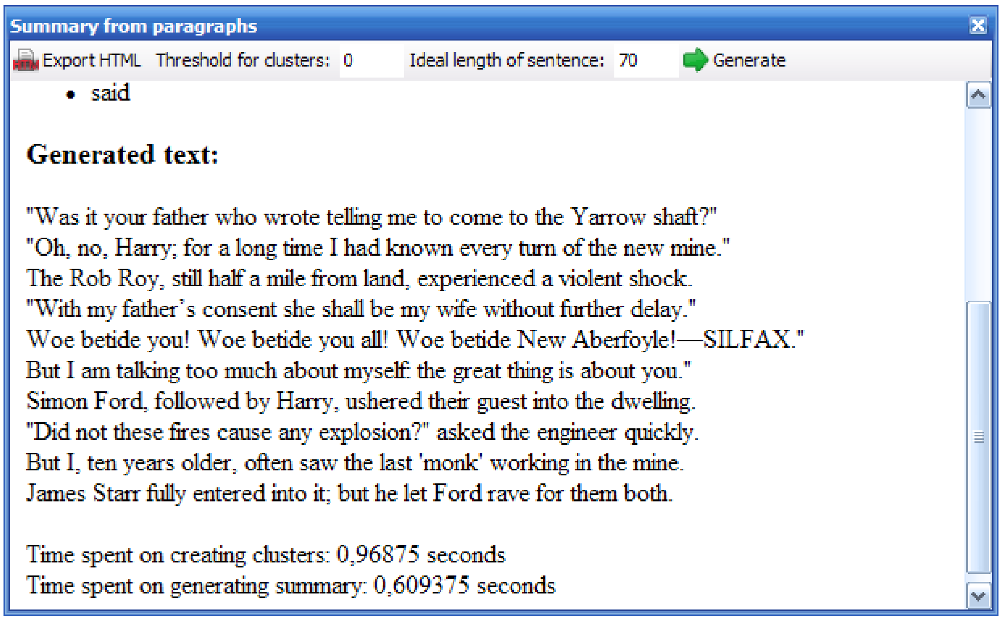
The user could also specify the threshold value in top menu, which is used to cut through the hierarchical clusters.

6.4.7. Generating text summary

For automated generating of text summary, there are three possibilities accessible from the document window:

- Summary from chapters – clusters on chapter level are used
- Summary from paragraphs – clusters on paragraph level are used
- Summary from sentences – clusters on sentence level are used

Figure 14 shows example of summary window. After pressing *Generate* button on top, it generates the text and also displays time spent of computation. The time spent on creating clusters shows the time spent on getting the clusters and in case of loading from cache, it could be significantly lower, than real time needed to compute them.



Threshold for clusters: 0
Ideal length of sentence: 70

- said

Generated text:

"Was it your father who wrote telling me to come to the Yarrow shaft?"
"Oh, no, Harry; for a long time I had known every turn of the new mine."
The Rob Roy, still half a mile from land, experienced a violent shock.
"With my father's consent she shall be my wife without further delay."
Woe betide you! Woe betide you all! Woe betide New Aberfoyle!—SILFAX."
But I am talking too much about myself: the great thing is about you."
Simon Ford, followed by Harry, ushered their guest into the dwelling.
"Did not these fires cause any explosion?" asked the engineer quickly.
But I, ten years older, often saw the last 'monk' working in the mine.
James Starr fully entered into it; but he let Ford rave for them both.

Time spent on creating clusters: 0,96875 seconds
Time spent on generating summary: 0,609375 seconds

Figure 14: Summary generating window

On this window, the user could set threshold and sentence length value. The threshold is used to cut through the hierarchical clusters. The ideal sentence length is used as a parameter for text generating, which is closely described in 2.3..

7. Programmer documentation

7.1. System requirements

The application requires Microsoft .Net Framework [7] in version 2.0 or later to be installed and Microsoft Visual Studio [11] 2008 or later to open the project solution.

For compiling the application, the System.Data.SQLite library [10] and Subversion [12] are required. The Subversion is used to automatically generate assembly version and it could be omitted by removing subwcrev command from pre-build events of SimDIS project.

For creating and compiling custom plugins it is not required to have the source and compile the whole application.

7.2. Compiling the whole application

The source code of application is contained within SimDIS solution of Microsoft Visual Studio. To open it, run SimDIS.sln in root of the solution.

The solution consists of several projects, of which the main are SimDIS and SimDIS.PluginInterface. Other projects are containers for plugins.

The whole application could be build by choosing *Build* > *Build solution* from the top menu.

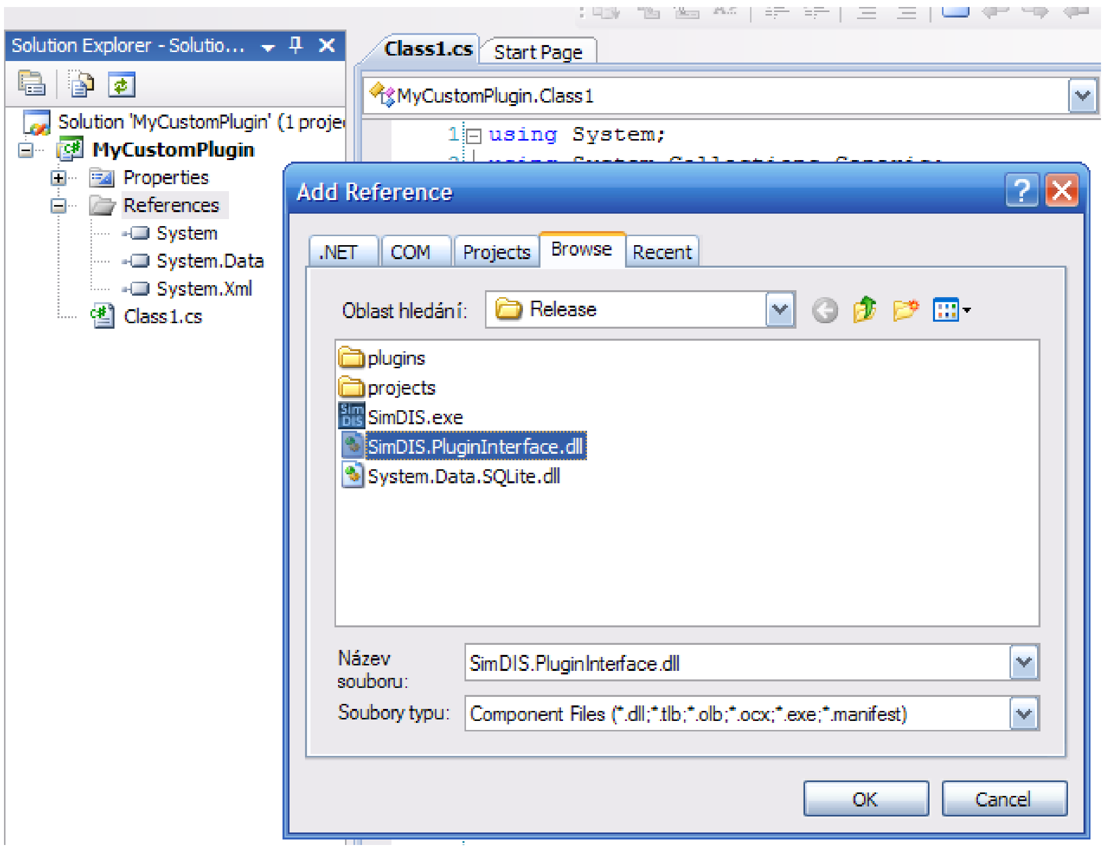
7.3. Custom plugin creation

The types of plugins are described in section 3.5.. This section describes creation of custom plugin without the need of the whole application using Microsoft Visual Studio 2008.

At first, the project (and solution) must be created. It is a Class library project and its name must end with Plugin. Then the reference to SimDIS.PluginInterface class library must be added as shown on Figure 15 and set the Copy Local property to false. Usually the System.Windows.Forms is also required to show visualization.

Next step it to rename Class1 to more suitable name, for example MyCustomVisualizer. It is important, that it ends correctly as described in section 3.5.. In this example I have chosen visualizer plugin, so the class must implement SimDIS.PluginInterface.IVisualizerPlugin interface.

The IVisualizerPlugin interface consists of getPluginName() method, which simply returns name of the plugin and visualize() method, which receives information about document and runs visualization.



```
using System;
```

Název souboru: SimDIS.PluginInterface.dll
Soubory typu: Component Files (*.dll;*.tlb;*.olb;*.ocx;*.exe;*.manifest)

Figure 15: Adding SimDIS.PluginInterface

A simple implementation of visualization is in Figure 16, where it shows the author and the title of provided document in System.Windows.Forms.MessageBox.

```
namespace MyCustomPlugin
{
        public class MyCustomVisualizer : IVisualizerPlugin
        {
                #region IVisualizerPlugin Members

                public string getPluginName()
                {
                        return "My custom";
                }

                public void visualize(IDocumentInfo documentInfo,
                        IWin32Window owner)
                {
                        MessageBox.Show(documentInfo.getDocument().Author
                                + ": " + documentInfo.getDocument().Title);
                }

                #endregion
        }
}
```

Figure 16: Implementation of simple visualization plugin

When building the plugin, it is important to chose right target platform. The SimDIS application is by default compiled for x86, so the plugins should be compiled also for x86 or Any CPU.

To add and run compiled DLL of plugin, it is only needed to copy in plugins directory of SimDIS application. The application will automatically detect and register the plugin.

Bibliography

[1] Manning Ch. D., Raghavan P., Schütze H.: *An Introduction to Information Retrieval*, Cambridge University Press, Cambridge, England, 2009

[2] Gan Guojun, Chaoqun Ma, Jianhong Wu: *Data Clustering: Theory, Algorithms, and Applications*, ASA-SIAM Series on Statistics and Applied Probability, SIAM, Philadelphia, ASA, Alexandria, VA, 2007

[3] Aone Ch., Larsen B.: *Fast and Effective Text Mining Using Linear-time Document Clustering*, KDD-99 San Diego CA USA, 1999

[4] Hinneburg A., Keim D.: *Optimal grid-clustering: Towards breaking the curse ofdimensionality in high-dimensional clustering.*, In Proceedings of the 25th internationalconference on very large data bases (VLDB ’99), San Francisco, 1999

[5] Bossard A.: *Generating Update Summaries : Using an Unsupervized Clustering Algorithm to Cluster Sentences*, Laboratoire d'Informatique de Paris-Nord, 2011

[6] World Wide Web Consortium (W3C): *W3C HTML*, http://www.w3.org/html/

[7] Microsoft Corporation: *Microsoft .Net Framework*, http://www.microsoft.com/net/

[8] Microsoft Corporation: *Windows Homepage*, http://windows.microsoft.com/

[9] SQLite development team: *SQLite Home Page*, http://www.sqlite.org/

[10] Simpson R., SQLite development team: *System.Data.SQLite*, http://sqlite.phxsoftware.com/

[11] Microsoft Corporation: *Microsoft Visual Studio*, http://www.microsoft.com/visualstudio/

[12] Apache Software Foundation: *Subversion*, http://subversion.apache.org/

CD contents

Enclosed CD contains:

- dp.pdf – A PDF version of this diploma thesis.
- Binary – Compiled version of the application
- Documentation – Generated documentation of the application
- Source – Source code of the application
- TestDocuments – A sample documents compatible with SimDIS application

List of tables

Appendix A: Generated summaries from K-means algorithm

K = 10 ; Required sentence length = 70

Automatically generated from chapter clusters:

Then, continuing, "And can you tell me what you father wants with me?"
"Are you still determined to explore this abyss?" whispered Jack Ryan.
The Rob Roy, still half a mile from land, experienced a violent shock.
"Decidedly, I have not your legs, my lad," said the engineer, panting.
"Harry," said Simon Ford, turning to his son, "light our safety lamps."
"Did not these fires cause any explosion?" asked the engineer quickly.
James Starr fully entered into it; but he let Ford rave for them both.
"Superb! Mr. Starr, superb!" answered Ford; "just look at it yourself!"
The inhabitants of Irvine would not have taken them there at any price.
THREE years after the events which have just been related, the guide–books recommended as a "great attraction," to the numerous tourists who roam over the county of Stirling, a visit of a few hours to the mines of New Aberfoyle.

Automatically generated from paragraph clusters:

"Was it your father who wrote telling me to come to the Yarrow shaft?"
Then, continuing, "And can you tell me what you father wants with me?"
"Are you still determined to explore this abyss?" whispered Jack Ryan.
They frequently met, either at the cottage or at the works in the pit.
"With my father's consent she shall be my wife without further delay."
Woe betide you! Woe betide you all! Woe betide New Aberfoyle!—SILFAX."
"Did not these fires cause any explosion?" asked the engineer quickly.
But I, ten years older, often saw the last 'monk' working in the mine.
James Starr fully entered into it; but he let Ford rave for them both.
The old overman stepped forward, and himself felt the schistous rock.

Automatically generated from sentence clusters:

Then, continuing, "And can you tell me what you father wants with me?"
"Are you still determined to explore this abyss?" whispered Jack Ryan.
It was like the sound of a mighty cataract rushing down into the mine.
They perceived at once that the waters of Loch Malcolm were rising.
Woe betide you! Woe betide you all! Woe betide New Aberfoyle!—SILFAX."
But I am talking too much about myself: the great thing is about you."
From the bottom of the Yarrow shaft radiated numerous empty galleries.
"Yes, indeed! I have the whole plan of the old pit still in my head."
"Did not these fires cause any explosion?" asked the engineer quickly.
James Starr fully entered into it; but he let Ford rave for them both.

Appendix B: Generated summaries from centroid HAC algorithm

Required sentence length = 70

Automatically generated from chapter clusters:
threshold = 0,51

"It's a long way off, is Edinburgh!" answered the man shaking his head.

"Was it your father who wrote telling me to come to the Yarrow shaft?"

They frequently met, either at the cottage or at the works in the pit.

Westward rose many hill–tops, soon to be illuminated by tips of fire.

The Rob Roy, still half a mile from land, experienced a violent shock.

Silfax, gazing upwards with wild and contracted features, appeared to become aware that the gas, lighter than the lower atmosphere, was accumulating far up under the dome; and at a sign from him the owl, seizing in its claw the lighted match, soared upwards to the vaulted roof, towards which the madman pointed with outstretched arm.

As to Simon Ford, the ex–overman of New Aberfoyle, he began to talk of celebrating his golden wedding, after fifty years of marriage with good old Madge, who liked the idea immensely herself.

If Jack Ryan and the other superstitious fellows in the mine had seen these lights, they would, without fail, have called them supernatural, but Harry did not dream of doing so, nor did his father.

"Did not these fires cause any explosion?" asked the engineer quickly.

James Starr fully entered into it; but he let Ford rave for them both.

THREE years after the events which have just been related, the guide–books recommended as a "great attraction," to the numerous tourists who roam over the county of Stirling, a visit of a few hours to the mines of New Aberfoyle.

Automatically generated from paragraph clusters:
threshold = 0,06

No signature.

Thus also shells, zoophytes, star–fish, polypi, spirifores, even fish and lizards brought by the water, left on the yet soft coal their exact likeness, "admirably taken off."

"Hullo, Jack! Where are you?"

A soft, transparent film of vapor lay along the horizon; the first sunbeam would dissipate it; to the maiden it exhibited that aspect of the sea which seems to blend it with the sky.

On this memorable occasion, Jack Ryan, in his favorite character of piper, and in all the glory of full dress, blew up his chanter, and astonished the company by the unheard of achievement of playing, singing, and dancing all at once.

"No doubt there would, Harry; it must be acknowledged, however, that nature has shown more forethought by forming our sphere principally of sandstone, limestone, and granite, which fire cannot consume."

"Some neighbor, then?"

But I, ten years older, often saw the last 'monk' working in the mine.

"Do they want to run ashore?" said another.

Automatically generated from sentence clusters:

threshold = 0,02

No signature.

He also wrote to excuse himself from two or three engagements which he had made for the week.

An obstacle speedily arrested his progress.

It was enchanting.

It was by this time three o'clock in the afternoon.

But, unluckily, this dog was lively, and barked.

"Some neighbor, then?"

It was the same with the "cockyleeky," a cock stewed with leeks, which merited high praise.

He was par excellence the type of a miner whose whole existence is indissolubly connected with that of his mine.

But I, ten years older, often saw the last 'monk' working in the mine.

It might be called a hive with numberless ranges of cells, capriciously arranged, but a hive on a vast scale, and which, instead of bees, might have lodged all the ichthyosauri, megatheriums, and pterodactyles of the geological epoch.

This vault served as a basement to Dumbarton.