



University of London

# 6CCS3PRJ Final Year Automated Timeline Extraction

Final Project Report

Author: Oliver Höhn

Supervisor: Dr Jeroen Keppens

Student ID: 1426248

March 18, 2017

## **Abstract**

The abstract is a very brief summary of the report's contents. It should be about half-a-page long. Somebody unfamiliar with your project should have a good idea of what your work is about by reading the abstract alone. -Summary of Project When legal and related professionals examine a case, they receive a substantial number of documents. These documents need to be examined in a useful manner to understand the events occurred. One useful perspective to understand what happened is a timeline of events. However, reading a large collection of documents and producing a timeline can be cumbersome. The aim is to ease this task by producing a system that can show timelines of events based on a set of documents provided.

### **Originality Avowal**

I verify that I am the sole author of this report, except where explicitly stated to the contrary.  
I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Oliver Höhn

March 18, 2017

## **Acknowledgements**

It is usual to thank those individuals who have provided particularly useful assistance, technical or otherwise, during your project. Your supervisor will obviously be pleased to be acknowledged as he or she will have invested quite a lot of time overseeing your progress.

-Acknowledge Supervisor, Friends & Family I would like to thank my supervisor, Dr. Jeroen Keppens. The supervision and support provided was extremely helpful and helped in the progression of the project. Also I would like to thank my family and friends for the continued support and encouragement throughout the project.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Project Scope . . . . .	3
1.2	Objectives . . . . .	3
1.3	Report Structure . . . . .	4
<b>2</b>	<b>Background</b>	<b>5</b>
2.1	Natural Language Processing . . . . .	5
2.2	Data Processing and Representation . . . . .	9
2.3	Normalizing Dates . . . . .	13
<b>3</b>	<b>Report Body</b>	<b>15</b>
3.1	Section Heading . . . . .	15
<b>4</b>	<b>Requirements &amp; Specification</b>	<b>16</b>
4.1	Brief . . . . .	16
4.2	Requirements . . . . .	16
4.3	Limitations . . . . .	18
4.4	Additional Aims . . . . .	19
<b>5</b>	<b>Design</b>	<b>20</b>
5.1	Objectives . . . . .	20
5.2	Use Cases . . . . .	21
5.3	Architecture . . . . .	24
5.4	Design Patterns . . . . .	32
5.5	UI . . . . .	33
<b>6</b>	<b>Implementation</b>	<b>37</b>
6.1	Approach . . . . .	37
6.2	Tools & Software Libraries . . . . .	38
6.3	Issues . . . . .	40
6.4	Testing . . . . .	48
6.5	UI . . . . .	49
6.6	Important Algorithms . . . . .	52
<b>7</b>	<b>Professional and Ethical Issues</b>	<b>58</b>
7.1	Section Heading . . . . .	58

<b>8 Results/Evaluation</b>	<b>59</b>
8.1 Software Testing . . . . .	59
8.2 Section Heading . . . . .	59
<b>9 Conclusion and Future Work</b>	<b>60</b>
Bibliography . . . . .	62
<b>A Extra Information</b>	<b>63</b>
A.1 Tables, proofs, graphs, test cases, ... . . . .	63
<b>B User Guide</b>	<b>64</b>
B.1 Instructions . . . . .	64
<b>C Source Code</b>	<b>65</b>
C.1 Instructions . . . . .	65

# Chapter 1

## Introduction

This is one of the most important components of the report. It should begin with a clear statement of what the project is about so that the nature and scope of the project can be understood by a lay reader. It should summarise everything that you set out to achieve, provide a clear summary of the project's background and relevance to other work, and give pointers to the remaining sections of the report, which will contain the bulk of the technical material. -What is Project About (incl scope)? What is Aim? Background? Relevance to other works? (Pointers to other sections?) The project aims to facilitate the understanding of a substantial number of documents, especially in law cases. When an employee is tasked with a law case, it is expected that they fully understand the overall structure and occurrence of events. However, when a large collection of documents are involved, this task can be both cumbersome for the employee and expensive (in time and financially) for the employer. Since time is spent reading and understanding the documents, instead of moving ahead with the task that the documents are used for.

### 1.1 Project Scope

Due to the system receiving as input a collection of documents, then processing and graphically showing a timeline, the main areas are Natural Language Processing (NLP), and Data Processing and Representation. All of which will be discussed in the Background section.

### 1.2 Objectives

//what is an event (summary, size, subjects, date, etc)

The Objectives are to produce a system that is simple to use and effective. This system should take in as input a selection of documents written in correct English (i.e. in natural language) and produce a timeline.

Since most users are from other fields, not Computer Science, they should be able to understand that the system requires as input documents and produces timelines. The timeline should self-explanatory, such that the user understands which events happened during certain dates, and what each event means. The system should produce responses in an appropriate time based on the input, and allow the user to rectify where the system has made mistakes. In addition, it should allow expert users to be able to change some of the input parameters used when the events are produced, such as the length of the summary or how many processors can the program use in parallel (to limit/improve the performance). As it will be likely that the users will want to save the produced timeline for later use, they will have the ability to choose between saving the timeline as a PDF or as a JSON. The latter allows for the system to be used with 3rd-parties that would like to change the graphical representation or manipulate the given timeline in their own system.

### **1.3 Report Structure**

In the following chapter, the background of the project will be presented in detail. Which is then followed by the design architecture and patterns used in the system. Followed by the implementation, testing and analysis of the system. In the analysis it will be determined how the requirements have been met. In the final chapter, the project will be concluded and improvements for future work will be discussed.



## Chapter 2

# Background

The background should set the project into context by motivating the subject matter and relating it to existing published work. The background will include a critical evaluation of the existing literature in the area in which your project work is based and should lead the reader to understand how your work is motivated by and related to existing work.

//explain what an event (from what is it built off) //defintion of an event

The resulting system aims to produce a timeline of events based on the input text. An event is given by its date(s), subjects and a short summary of the sentence that produced it. An event is produced when a given sentence contains a date. An event can have more than 1 date if it is considered to happen in a range of dates. For example, an event that happened in the 1980s would have two dates, one for the start date: 1980-01-01, and one for the end date: 1989-12-31. While an event that happened just on one day would have just one date. The subjects of an event are given by the "person, place, thing, or idea that is doing or being something"[1].

### 2.1 Natural Language Processing

-explain what NLP is

The projects primary area of research is Natural Language Processing (NLP). NLP is the area of computer science where the aim is to translate human readable and spoken language to a computer (cite). This requires the human input to be subject to constraints. Thereby in this project, expection on the input text are assumed. For example, it is to be expected that documents are written in correct English. As every language has their own grammar, and

thereby, their own rules, to expand the system to different languages would require different rule sets to be applied depending on the language to process the text. This includes the algorithms used in the summary of sentences used in events (discussed in a latter section).

A relevant issue is the input documents having text that is disorganised in a grammatical sense. Many NLP software tools (including StanfordCoreNLP used in this project) perform extremely poorly with such input text. For example, in the paper Named Entity Recognition in Tweets: An Experimental Study [7], where they looked at the performance of popular NLP tools on "Tweets", which due to them being limited to a character count will use abbreviations that do not make sense grammatically. This is due to the fact that the NLP tools and algorithms cannot apply their rules and models to the text to identify the different components. Thereby, in this project the assumption is that the input will be in correct English, as the systems primary user is a law professional. NLP will be presented in more detail in the following section. -what parts of NLP are involved

NLP is a broad area of study. However, in this project the focus is on Automatic Summarization, Named-Entity Recognition (NER), and Sentence Breaking.

In Automatic Summarization the aim is to produce a shorter version (the summary) of a given input text, that still holds the same meaning of the original input. The summary can be built directly from the words in the input, or it can be built using a dictionary. As in this project, an event is built from one sentence that contains a date, the specific area of Automatic Summarization which was focused on was Headline Generation. This is where a summary is built based on a given input text, such that the summary falls below a certain threshold value. For Headline Generation there are two main implementations: statistic based and decision (or trimming) based [3].

In the statistic based model, where Noisy-Channel models are the most prominent, as shown by the multitude of publications [2, 3, 8]. In noisy-channel models, the belief is that the summary of the given input lies within the text but it is surrounded by unwanted noise (text). These systems require a large collection of annotated data (pairs of input and their summary), which is used in the calculation of the statistics of whether or not a produced summary correctly represents the input. Examples of these algorithms can be found in the works of [3, 5].

The decision based models, are older than the statistic based models and use the grammar of the input text to trim (remove) parts of the inputs until no more rules can be applied or the summary produced is below a given threshold [4]. This is done by tokenizing, breaking an input text into words, phrases, symbols and tagging them each by an identifier(cite). The tokenized

text, which is usually represented as a tree where the leaves are the words in the input text and the inner children the identifiers, is passed through an algorithm which applies rules which removes branches of the tree until no more rules can be applied or the summary text is below a given threshold. The trimmed tree is then used to produce the summary. //compare statistics to decision

The trimming based models do not tend to produce as good of summaries as the statistic based model, due to them producing, usually, only one summary while the statistic based models produce a selection to choose from. However, as can be seen from the works of Knight and Marcu [5], the trimming based models can produce better summaries than the statistic models in some occasions. The main advantage of the trimming model is their speed, and not requiring a large corpus of data (like the statistic models) by relying on the grammar to build the summary. In newer works of text summarization, neural network models are being used. These fall under the statistical based models. They produce extremely accurate results, but as most statistical models they require a large corpus of data. They requirement of the extensive sample of annotated data also leads to these algorithms being processor-heavy, as the data needs to be read, and calculations need to be performed to produce the probability values used in identifying suitable summaries.

Due to the time-constraints of the project, it was decided to use the trimming approach, in specific the algorithm provided by Dorr, Zajic, and Schwartz [4] (Figures 1 and 2). Where the given input text is turned into a tree based on its grammatical structure, with the words in the text as the leaves, and the inner nodes being the identifiers, which is then trimmed. In addition, in statistical models the input text size is multiple sentences, i.e. a paragraph or more of text, where loading the models from the annotated data has less of an impact in performance as this being done for every sentence in that input text. -cite

---

**Algorithm 1:** Dorr, B., Zajic, D. and Schwartzm R, (2003). Hedge Trimmer: A Parse-and-Trim Approach to Headline Generation

---

**Input** : A Grammatical Tree T of the Sentence to summarize

**Input** : threshold: Threshold value

**Output:** A Summary of the given sentence

```
1 get the leftmost-lower subtree with root S;
2 remove time expressions;
3 remove determiners (e.g. 'a', 'the');
4 while the number of leaves in tree > threshold and there are subtrees to remove with this
   rule do
5   | remove all the children except the first, where the rightmost-lowest subtree with root
   |   XP and its first child also being an identifier XP (where XP can be NP, VP, or S);
6 end
7 while the number of leaves in tree > threshold and there are subtrees to remove with this
   rule do
8   | remove any XP (where XP can be NP,VP,PP) before the first NP found;
9 end
10 get Tree T' from lastRule;
11 from T' create a sentence S by reading of the leaves in pre-order;
12 return S;
```

---

---

**Algorithm 2:** Last Rule

---

**Input** : A Grammatical Tree T of the Sentence to summarize

**Input** : threshold: Threshold value

**Output:** A Grammatical Tree of the Summary of the input after the last rule has been applied

```
1 if the number of leaves in tree  $\geq$  threshold then
2   make a copy of the tree T';
3   while the number of leaves in tree  $>$  threshold and there are subtrees to remove with
      this rule do
4     remove any trailing PP nodes (and their children);
5   end
6   if the number of leaves in tree  $>$  threshold then
7     while the number of leaves in tree  $>$  threshold and there are subtrees to remove
        with this rule do
8       remove any trailing SBARs (and their children);
9     end
10    while the number of leaves in tree  $>$  threshold and there are subtrees to remove
        with this rule do
11      remove any trailing PPs (and their children);
12    end
13  end
14  return T';
15 end
16 return T;
```

---

## 2.2 Data Processing and Representation

In the grammatical tree formed, the inner leaf identifiers are given by the P.O.S Treebank<sup>1</sup>. Each word or set of words are given a part of speech tag identifying them. An example can be found below. (example figure).

On the grammatical tree shown in the previous figure, we will apply the algorithm proposed by Dorr, Zajic, and Schwartz [4]. This can be shown graphically in the following figure. It is to be noted that the value of the threshold does not force the summary to be below it, but it is

---

<sup>1</sup>[http://www.ling.upenn.edu/courses/Fall\\_2003/ling001/penn\\_treebank\\_pos.html](http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html)

a length of the summary to which the algorithm is working to. For the sentence: "On Friday the Washington Post came out with the latest from its long-running investigation into Trump's charitable donations.", the following gramantical tree is produced 2.1.



Figure 2.1: P.O.S/Grammatical Tree of: "On Friday the Washington Post came out with the latest from its long-running investigation into Trump's charitable donations."

The inner nodes in the tree are identifiers given by the P.O.S Treebank, and the leaves are the words in the sentence. A parent identifier can be a broader identifier of a collection of sub-identifiers. Using the algorithm described previously [4], the tree can be processed as follows. The following example is used as a visualisation of how the decision-based algorithm works. Firstly, the lowest-leftmost S must be identified, as can be seen from the figure 2.1 there is only one subtree with root S. This subtree is extracted, and the algorithm is continued on it. The next step is to remove time expressions, of which there is only one "Friday", however we must remove its parent to (and thereby its children) to avoid producing a grammatically incorrect summary. The result is graphically shown in the following figure 2.2.



Figure 2.2: P.O.S/Grammatical Tree after removing time expressions

On the resulting tree, the algorithm dictates the removal of some determiners. A determiner is identified by the "DT" parent tag, however not all of the parents identified are removed, only ones that have a child of label "the" or "a". The resulting tree of applying this rule is given by

Figure 2.3.



Figure 2.3: P.O.S/Grammatical Tree after removing determiners

The threshold value is used in the next following rules. For this example, the threshold value is 10. From the two rules that are left, only one will be applied as the resulting tree's number of leaves will fall below the threshold value, such that the last rule will not be applied to avoid the algorithm from trimming the summary to much. Hence, the threshold's value importance is of stopping the algorithm from over-trimming or under-trimming. When the tree is over-trimmed there is the possibility that the resulting summary does not have the core meaning of the original sentence, or has no meaning. The advantage of under-trimming is that the meaning of the sentence is very likely kept, due to the summary sentence being of greater length of the optimal summary, thereby having more words and such more meaning, and being closer to the original sentence. However, the aim is to produce a summary, i.e. a short sentence with the same meaning as the original, thus under-trimming can ensure the meaning is kept in the summary, but not that the resulting summary is optimal in size or time (when reading many summaries, shorter ones will be appreciated over longer ones). When applying the XP-Over-XP rule, when a parent of identifier type XP (where XP can be NP,VP or S), has a first child identifier of type XP also, then all other children of the parent are removed. This is done iteratively until the resulting tree is below the threshold, or the rule cannot be applied further due to there not being anymore parent XP, first child XP pairs. The first iteration of the rule is shown in Figure 2.4, and the second and final iteration in Figure 2.5.

If the number of leaves did not fall under the threshold value, then the next rule to be applied is XP-Before-NP. In this rule, any XP before the NP of the sentence (the grammatical subject) is removed. This would be carried out until the number of leaves falls below the threshold, or the rule cannot be applied further. Finally, the last rule is applied where iteratively trailing PP and SBAR subtrees are removed until the threshold is reached or the rule cant be applied. The result



Figure 2.4: P.O.S/Grammatical Tree after XP-Over-XP first iteration

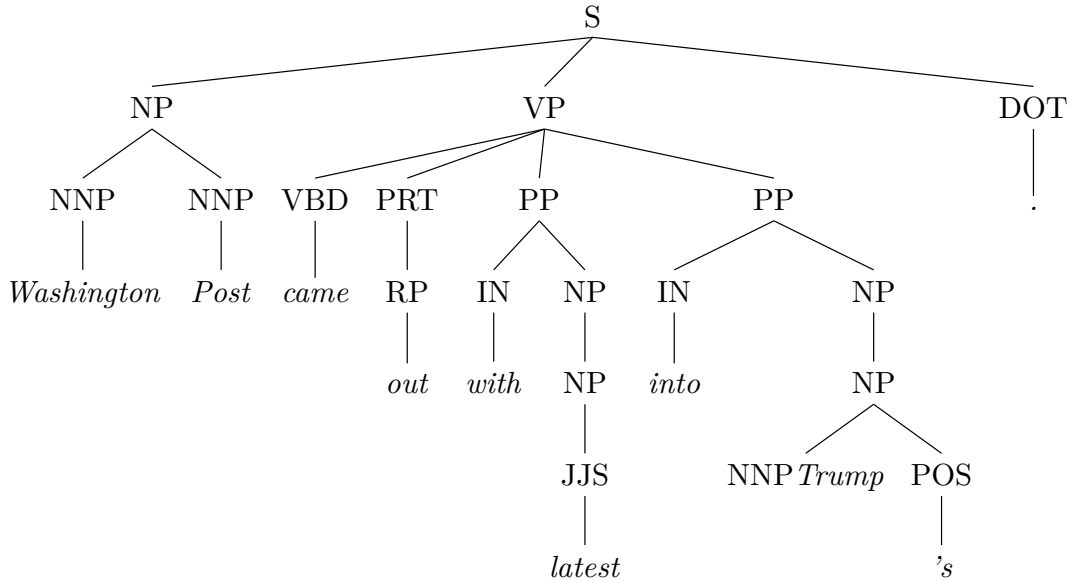


Figure 2.5: P.O.S/Grammatical Tree after XP-Over-XP second iteration

of applying this algorithm (ref algorithm) on the input text: "On Friday the Washington Post came out with the latest from its long-running investigation into Trump's charitable donations.", is "Washington Post came out with latest into Trump's". It should be noted that the original meaning of the sentence, which is that the Washington Post released a new article, is kept. However, it should also be noted that the resulting summary is not grammatically correct as "Trump's" should be "Trump". Even though the summary is not grammatically correct the summary is significantly shorter than the input text, and does represent the same event. It is a headline of the input, and should indicate to a reader what the event is about, to allow them to have a general understanding of what occurred. -tags are POS (cite) //d -example of input text to tree //d -example of producing summary //d -algorithm //d -what are the options for



the summary (Neural Networks vs Decision-Based) //d -give an algorithm for determining the summary, with an example //d

-explain the date problem, with example (determine that it uses an ISO standard)

## 2.3 Normalizing Dates

An issue when processing the documents and picking out events, and then placing them in a timeline, is that it is very likely that sentences will not include the full date of when an event occurred. Due to how a temporal expression such as "Yesterday" or "Last week" have different meanings depending on the context in which they are in, it is necessary to determine an exact date for that event to be able to produce a timeline. For example, if the text was written on the 11th of March 2017, then "Yesterday" in its context to the 10th of March 2017, but if the document was written on the 1th of February 1689, then it refers to the 31st of January 1689. Thereby, it is important to be able to infer reliably and accurately the date to which ambiguous time expressions refer. Since the dates will be needed to compare the events in a timeline, and be able to sort them. Thereby imposing constraints, where events that happens within the same time period are grouped together and not separated by events that happen it completely different time eras. This is especially valuable to the user, as they should be able to see events that happen within the same time period close together not separated.

To determine the context of where an event occurred, it will be necessary to have a reference point, a base date. In other timeline works, similar techniques have been used [6]. The reference point should be the context in which the document was written in, or was aimed to be written in. This can be the publishing date of an article, or the creation date of a document, or any date which allows the exact date to be determined. It should be noted that for exact dates that are described in text, such as "On the 12th of December 1996...", the reference point has no value, as it can be determined without it that this text refers to an event that occurred on 12-12-1996.

There is also the possibility that temporal expressions point to a range of dates, i.e. "In the 1980s...". While it is not possible to determine the exact date, or dates, this event is describing, it can be determined reasonably [6] that it is somewhere between the start of 1980s, i.e. 01-01-1980, and the end of the 1980s, i.e. 31-12-1989. This can be used to attempt to provide exact dates for that event, a start and end date, to then compare with other events in the production of the timeline. This approach of determining start and end dates where events could have occurred, is used in the implementation of the system. While it might not tell the user exactly

when the event occurred, it gives them an idea of when it could have, which is the system's aim.

## Chapter 3

# Report Body

The central part of the report usually consists of three or four chapters detailing the technical work undertaken during the project. **The structure of these chapters is highly project dependent.** They can reflect the chronological development of the project, e.g. design, implementation, experimentation, optimisation, evaluation, etc (although this is not always the best approach). However you choose to structure this part of the report, you should make it clear how you arrived at your chosen approach in preference to other alternatives. In terms of the software that you produce, you should describe and justify the design of your programs at some high level, e.g. using OMT, Z, VDL, etc., and you should document any interesting problems with, or features of, your implementation. Integration and testing are also important to discuss in some cases. You may include fragments of your source code in the main body of the report to illustrate points; the full source code is included in an appendix to your written report. -Tasks in project (Design, Implementation, Experimentation, Optimissatio, Evaluation) -present alternatives, compare them, why picked -justify software used -problems identified -important features -testing -stanford corenlp pos tags

### 3.1 Section Heading

#### 3.1.1 Subsection Heading

## Chapter 4

# Requirements & Specification

### 4.1 Brief

-purpose of project -how established requirements

The system should take as input a set of documents, process them autonomously (i.e. without the users involvement), and produce a graphical representation of events in the documents. This requires the identification of sentences in the text that contain dates, provide an exact date (to compare to other events), identify subjects and provide a summary of the sentence. From this the requirements can be determined.

### 4.2 Requirements

#### 4.2.1 Functional Requirements

The functional requirements of a system are behaviours a system should have<sup>1</sup>. In this project the functional requirements are as follows:

1. Process documents of different file types (e.g. .pdf, .txt, and .docx).
2. Identify dates and subjects in text.
3. Summarize sentences.
4. Produce a graphical timeline of the events in the input documents.
5. Modify/Delete events in the timeline.

---

<sup>1</sup><http://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/>

6. Travel between timeline and relevant document.
7. Save timeline (as .pdf or .JSON).

As the software will require as input text, the 3 most used document file types<sup>2</sup> should be processable. As the aim is to identify events in the input text, it is trivial that dates should be identified. The subjects of a sentence are required to give an overall description of what the event is about. Subjects, in this case, include names of people, locations, and quantities of money. The resulting system should enable users to modify events as it can be the case that the summary, subjects, or date determined by the system are wrong. Thereby allowing the user to correct the mistake.

The final two requirements do not affect the processing of the system, but can be advantageous to users. As being able to switch between timeline and document will provide the ability to go from a general description of an event to the actual, full-detail, and in context description. Saving the timeline to a PDF file will allow the results to be included in documents. However, more interestingly, producing an intermediate JSON output makes the system compatible with 3rd- applications that can provide other graphical representations and/or process further the data.

#### 4.2.2 Non-Functional Requirements

Non-functional requirements of a project are descriptions of how the system must do the functional requirements, and the qualities the system should have. In this project the non-functional requirements are as follows:

1. A responsive and intuitive UI (Visibility).
2. Reasonable output time (Efficient).
3. Identify the majority of events (Effective).

As the system should be used by any kind of user, with any technical knowledge, it should be understandable for the user how to use it, i.e. the system should be usable. The users of the system will be described further in later chapters. It is trivial that the system should produce an output reasonable to the amount of input given. It would be unreasonable that the resulting system should have an exponential running time, i.e. given an input of amount  $n$ , the system carries out more than  $2^n$ , this would be  $O(n^2)$ . The system should be efficient. Identifying

---

<sup>2</sup><http://www.computerhope.com/issues/ch001789.htm>

most events is the most important non-functional requirement, and one of the most important general requirements of the system. The system should be able to extract simple events in text where the full date is mentioned, but also be able to extract more complicated events where the temporal expressions are ambiguous. Thus, making the system effective.

A point that will be further explored in later chapters, is the extraction, and inference to concrete dates, of ambiguous temporal expressions, which are known to have happened after and/or before other temporal expressions, which can be ambiguous or concrete. This would produce a timeline, where the exact date of when an event occurred is not known, however it is known that it happened before or after another event, and thereby the timeline can be formed. However, this requires changing models in already established NLP tools, which is discussed further in the Future Works chapter.

### 4.3 Limitations

The greatest limitation of the project is time, both in its development and in the execution of the system. As the project time is limited, compromises have to be made. For example, a noisy-channel neural-network summary system would produce different plausible summaries for a given text, of which one should be a reasonable summary. However, as mentioned previously, noisy-channel models require a large amount of annotated data (and thus are domain-dependent). Thereby, providing this set of data would require more development time. In addition, the loading of these large models of data to summarize one sentence, would have a substantial impact in the running time of the system. If the system then takes longer than a user to produce the timeline, then the user will most likely prefer to produce the timeline manually.

A limitation that must be recognised, is the understanding of context. Natural-Language Processing (NLP) is still an area of research, and will continue to be. It is a clear issue in this project, that some texts will produce poor timelines, as the context in which the text was written in will not be encapsulated fully. This can be applied to ambiguous temporal expressions, or the meaning of the sentence. Some NLP tools attempt to maintain the context of text, by looking for references between sentences. However, this does not replace the context that humans understand when reading text or having a conversation.

## 4.4 Additional Aims

-open source (with documentation), to be used in 3rd parties and allow the project to be further developed

The project will be open-source. It will be available on GitHub, along with a license that allows anyone that is interested in processing text to use it. This will be beneficial, as the application does produce an intermediate JSON to be used by 3rd-party systems. Thereby, iteratively improving the system. Open-source may also be required, however this is dependent on the libraries used in the development.

# Chapter 5

## Design

-objectives -use cases (actors) -architecture of backend -design patterns -gui (with screenshots)

### 5.1 Objectives

**Visibility** - It should be visible to the user what are the functions that are available. This includes being able to distinguish actions from informative text. This can be ensured through the use of buttons that are highlighted. This would allow to provide an application that can be picked up and used with minimal to none training. Therefore, it would be beneficial to the growth of the systems use, by attracting more users through its simplicity of use. This can be completed by providing a simple and intuitive User Interface (UI), where buttons are highlighted, and the interface is not cluttered with actions, instead just providing what is necessary. The timeline produced will be available in three different formats: graphically in the system, as a PDF file, or as a JSON. The latter is the only one that is not a graphical representation, while the other two do.

**Efficient** - The time spent to perform tasks should be reasonable in the context of the input. This can be further expressed in mathematical notation of Big-Oh. If an algorithm takes an input of size  $n$ , and roughly performs  $n$  operations to produce a result, then the algorithm is said to have a runtime of Big-Oh of  $n$ . This allows the efficiency of an algorithm to be compared to other algorithms, and for the run-time to be scaled to larger inputs. An objective would be to avoid having a run-time, a Big-Oh, that is exponential. Since for a small  $n$  the run-time is large, therefore, for a large  $n$  it is infeasible for a result to be produced in time. This objective can be completed through the use of Threads. A Thread is a lightweight processor computation



unit. Using more than one-thread allows for tasks to be carried out in parallel. Therefore, if the input is  $n$  documents, and there are  $n$  threads, then the running time of the system would be the greatest running time of all the documents being carried out. Since if all documents are being processed in parallel, and completely independently of each other, the one document that takes the longest to be processed will give the time of processing all of the documents (cite). The amount of threads running in parallel should be an editable setting to the user, as they may wish to reduce the load of running many threads in parallel in the case they are doing other work on their machine, or they may wish to use the maximum amount of possible threads they can.

**Effective** - The system should meet its purpose. That is, its task is to take as input documents and produce a timeline. Thereby, the system should provide the user options to load documents, and then provide a graphical response. In the case where no response can be produced, i.e. due to the document encoding not being parsable, then the system should not attempt indefinitely to produce a timeline with that document, and instead produce an empty timeline.

## 5.2 Use Cases

A use case is a task a actor in the system may want to perform. An actor is any type of user of the system. In this case, the user can be a law professional that requires to have a general understanding of a given set law-related documents. Therefore, it can be assumed that the user does not necessarily have experience with NLP, and the tasks that are involved in processing the document. It should be transparent to the user how the documents are being parsed, and only if they are interested would they require to look at the available source-code. The technical skill of the user does not need to be of an expert, as the tasks required are to provide documents, and then from the resulting timeline they can traverse it and perform their analysis. In some cases, the user may produce their own graphical representation of a timeline and just use the produced JSON of the system.

The use cases of the system are given by the requirements, and they are found below. //use case stick figure diagram

1. Load Documents

- (a) Primary Actor: User

(b) Goal: load set of given documents, where the document file types can be .pdf, .docx, or .txt.

(c) Main Sequence:

- i. User selects the "Load Documents" option.
- ii. System prompts a File Selector.
- iii. User selects set of documents and the base dates (or reference dates) to use with them.
- iv. System responds with timeline of events.

## 2. Swap from Timeline to Document

(a) Primary Actor: User

(b) Goal: show the sentence, in context, that produced the given event.

(c) Main Sequence:

- i. User selects event.
- ii. System responds with dialog to "Edit Event" or "Go to Document".
- iii. User selects "Go to Document" option.
- iv. System opens new window with the text of the document where the event originates from, with the sentence that produced it highlighted.

## 3. Edit Event

(a) Primary Actor: User

(b) Goal: modify the data of an event.

(c) Main Sequence:

- i. User selects event.
- ii. System responds with dialog to "Edit Event" or "Go to Document".
- iii. User selects "Edit Event" option.
- iv. System responds with dialog with the data of the event set in fields.
- v. User edits the data as needed.
- vi. System validates the entered data, and saves.

## 4. Save Timeline

(a) Primary Actor: User

(b) Goal: save the produced timeline as a PDF or JSON.

(c) Main Sequence:

- i. User selects "Save To..." option.
- ii. System responds with option dialog to select the file format to save.
- iii. User selects the needed file format.
- iv. System responds with File Selector.
- v. User selects the location to save the timeline.
- vi. System generates the required data to save the timeline in the desired format and attempts to save it in the system.

//note load documents use case includes adding to an existing timeline, user is the person using the system, edit event incl delete, checks for invalid data, file not available

The main sequence are the steps of the interaction in the use case to reach the desired goal. An error can occur during the interaction. It should be the systems responsibility to deal with the error, and not end the execution of the program. The primary actor, is the agent, or entity that initiates the use case (cite or footnote).

It should be noted that loading a document includes both when it is the first set of documents to be loaded, i.e. the timeline is empty, and when there is already a populated timeline. In the latter case, it would be beneficial to discard duplicated events. A duplicated event is one where it is produced from the same file, using the same reference date, and the data is equal. This is done to not clutter the timeline with events that are repeated, as the timeline should be efficient in the data it provides, i.e. describe the events in the document with as little as possible of additional data. It can occur that two documents produce the same event, these should not count as duplicated, as the event may have a different context depending on what file it is from. The reference date is included, as for the same document different events may be produced depending on the reference date used for that document.

When an event is being edited, the user should have the option to delete said event. It can be that the system produced an event that is not relevant for the user, or another event describes it. This can be the case when a set of closely related documents are loaded, and two or more events have the same meaning, but are not direct duplicates as they originate from separate files. In addition, when the user edits the data of an event, validation checks are performed before the changes are saved. It can occur that the modification inserted invalid data. For example, if the date of an event is modified but instead of a new date being set, text

is placed. In the case of the event occurring during a range of dates, i.e. it has a start date and end date, a plausible validation check should be that the second date does not occur before the first. The system can do these checks before it attempts to save the modifications, and in cases where the validations fails, the changes should not be saved, instead the user should be prompted to enter correct data or cancel the action. The system should inform the user of where they validation failed, to allow them to correct the data.

It can occur that the desired location of where the timeline should be saved is unavailable. This can be due to the system not having write access to that part of the Operating System, or the file which is being overwritten is in use (i.e. another process locked the file (cite or footnote)).

## 5.3 Architecture

The architecture of a system, is the structure of the components of the system<sup>1</sup>. It focuses on the back-end, or logic, of the system as opposed to the graphical, or front-end task. A good software architecture is one where the components of the system are encapsulated with other related components. For this project, the software architecture has been produced using Unified-Modelling Language (UML). In UML, components or classes are represented by a rectangle, with their available functions listed. The architecture of the whole system is presented in the figure below (Figure 5.1). We will look at each package individually.

---

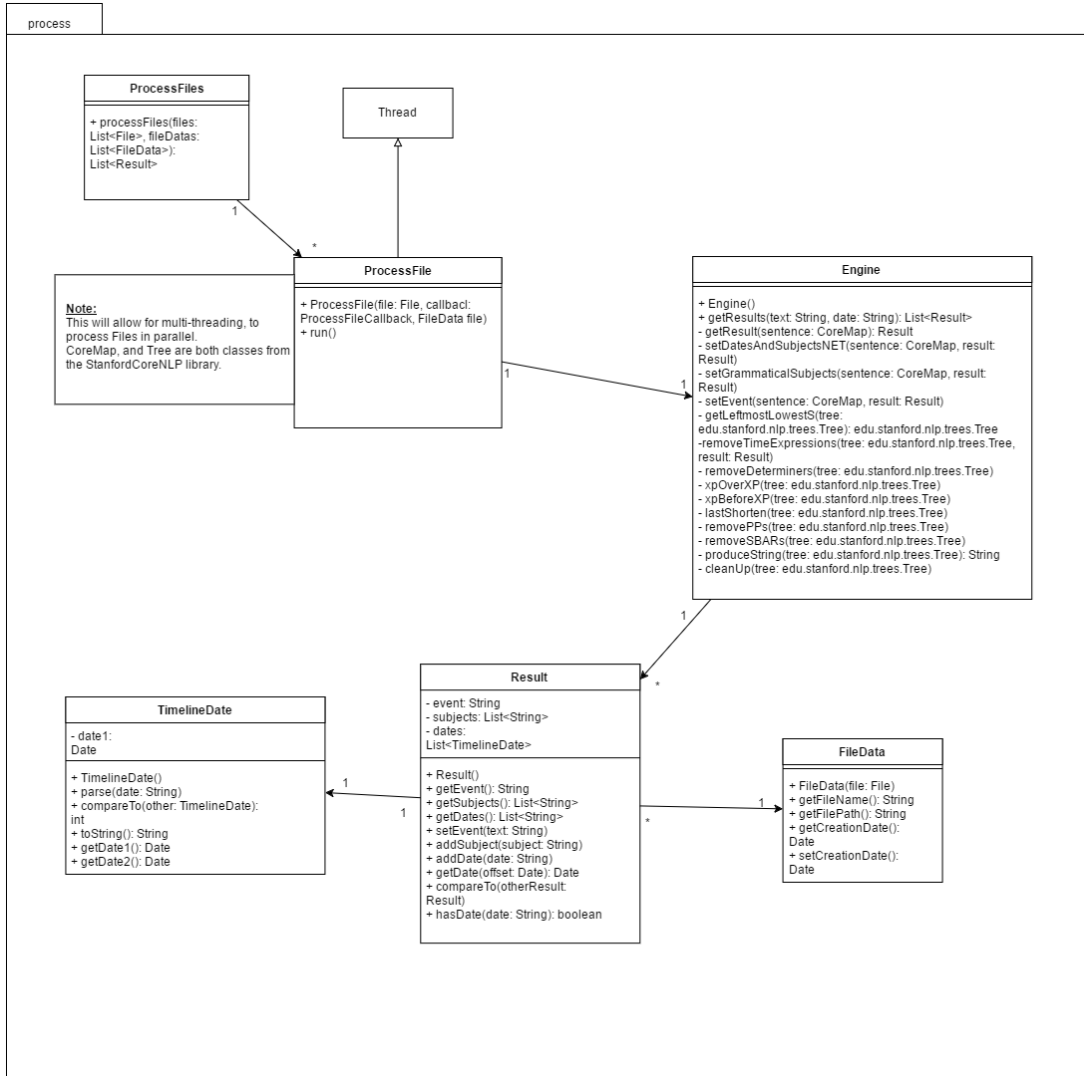
<sup>1</sup><https://msdn.microsoft.com/en-gb/library/ee658098.aspx>

[illegible]

The core of the system is the process package (see Figure 5.2). The architecture used here, is that interaction is done through ProcessFiles. A list of files is passed in, along with their data (such as the file name, its path, or the creation date used). Each file is processed in parallel. The maximum number of parallel processing that can be done is given by the settings of the system (in the settings package). This is the maximum number of threads that can be ran at any given point. However, in the implemented system one more thread should be added to the count, as the graphical user interface always runs on a separate thread. The idea is that if the maximum setting is  $n$ , then at any given point at most  $n$  files are being processed. Whenever one file finishes processing, another begins to be processed. As mentioned before, if  $n$  files are allowed to process at any given point, and the input size of documents is  $n$ , then the time it takes for the system to process all the documents is given by the greatest maximum time to process one of the  $n$  files. The pseudo-code for this is given below (see Algorithm 3), in an actual implementation semaphores can be used. A semaphore is a control data structure to limit how many processes can run in parallel. It is done by acquiring a lock when a process needs to be ran, if a lock is available then the process can run, if no lock is available then the process waits until one is available.

25

Figure 5.2: UML of the Processing Package



TimelineDate component.

When a temporal expression is detected in a sentence, it must be processed to an exact date to be useful. This is done through TimelineDate. It attempts to parse them. In this system, the decision was made to use the StanfordCoreNLP suite <sup>2</sup>. It is a well-known and tested tool for Natural Language Processing. Other tools exist, such as ApacheNLP, however Stanford's tool has a larger set of documentation and support, as well as being a thread-safe and efficient tool. Thread-safe refers to the ability to share this tool between separate processes without having to worry about concurrency issues.

When a file is processed, its text is extracted, and this is then passed through the Engine. The Engine is responsible for producing the Results for that file. It identifies sentences with

<sup>2</sup><http://stanfordnlp.github.io/CoreNLP/>

---

**Algorithm 3:** Algorithm for processing a list of Files

---

**Input** : A list of Files to Process

**Output:** A list of Results

```
1 foreach File in the input list do
2   wait until can run;
   /* if the maximum number of threads running in parallel has not been
      reached then stop waiting, else wait */
3   process the file;
4   add the produced Result to the list of Results to return;
5 end
6 return list of Results;
```

---

dates, and for those it extracts subjects such as people, locations, money, etc. In addition, it implements the Hedge-Trimmer discussed in the Background Chapter.

### 5.3.2 System Package

//talk about system to be shared throughout the system (same data shared)

The system package holds the system and settings components (see Figure 5.3). It is responsible for providing global settings, such as the maximum number of threads to be ran in parallel, the threshold value used in the sumamry algorithm, and graphical settings such as the width and height of the window. An interesting aspect of this package is the SystemState. As the system moves from start, to processing files and then finishing the processing. This can be modelled in the system. When files are passed through the process package, the state of the system is changed. This allows for the graphical representation of the system to be decoupled of the logic, as it can use the system state to determine when certain actions are available, when a loading bar needs to be shown, and when a timeline should be displayed. This component is shared throughout the system, as it contains data that is required in other components, both for the logical aspects of the system and the graphical aspects.

Figure 5.3: UML of the System Package



### 5.3.3 Ranges Package

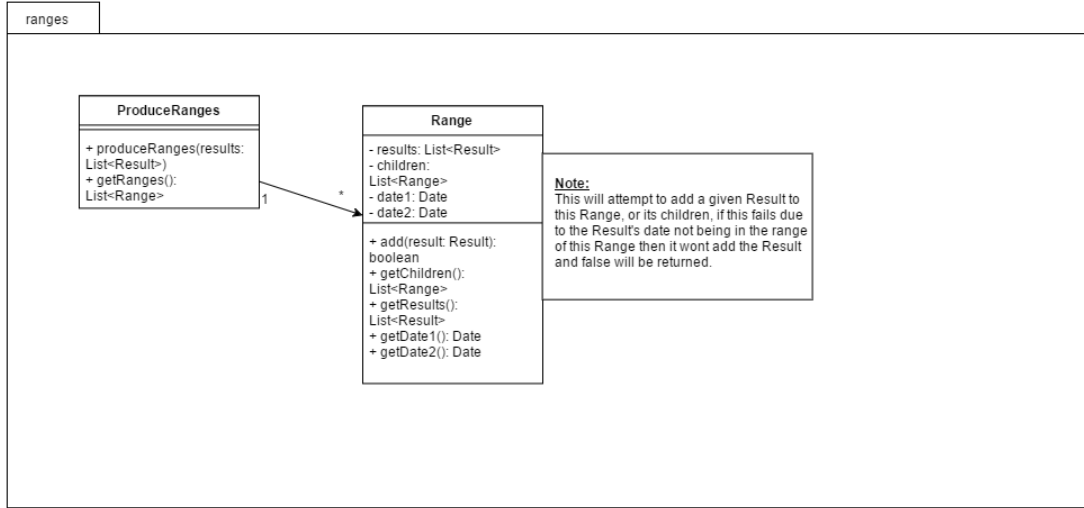
//trees of ranges with results, algorithm, example, running time

The ranges package was one of the later packages developed (see Figure 5.4). Its focus is on placing already produced Results into ranges of dates (see the Algorithm 4). A Range is



defined by a start and end date. A Result is placed within this Range if it has the exact same start and end date, if this is not the case then it attempts to look at the children of that Range. If it is not possible to place the Result in a Range or its children then another root Range is checked. If after checking all Ranges this is still not possible, then a new separate root Range is created using the Results dates and the Result is placed there. This allows to order events within each other, such that if one event occurred during the period of a longer event, than that event will be contained by the other.

Figure 5.4: UML of the Ranges Package




---

**Algorithm 4:** Algorithm for placing Results in Ranges

---

**Input :** A list of Results

**Output:** A list of Range roots, i.e. a forest of Ranges

- 1 sort the list of Results by the number of days in between the start and end date in descending order;
  - 2 **foreach** *Result in the sorted list* **do**
  - 3     attempt to add it to one of the existing Range roots;
  - 4     **if** *failed to add to existing Range* **then**
  - 5         make a new Range using the data of the Result;
  - 6         add the new Range to the list of Range roots;
  - 7     **end**
  - 8 **end**
  - 9 return list of Range roots;
- 

The Results are sorted by the range of the start and end date, i.e. the number of days between the two. A Result that only has one date, because it just occurred on that specific date, has a range of 0. The Results with the largest ranges are added first as it is more likely that in the possible dates in between their start and end date, there will be Results that have start and end dates there. This leads to the production of a tree, which will be shown below.

Where the root is a Range with a start and end date that encapsulates all the start and end dates that are in the tree. In some cases it will be necessary to expand the dates of a Range if it is the case that a Result has an event that overlaps with the dates of the Range. This will then lead to have a Range that has no results, but has two children, the newly made Range for the Result that is being added, and a subtree which was Range that was being overlapped.

For example, if you have the two Results presented in the table 5.5 and the pre-existing tree in Figure 5.6. The Results would be sorted by their range, such that the second result would be the first to be added. This would produce the tree in Figure 5.7. Afterwards, the first Result in the table would be added, producing the tree in Figure 5.8. In the Figure 5.8, the following occurred: it was determined that the Result that was being added overlapped an existing Range, so a new Range was formed that would encapsulate the previous Range and the new Result by extending the start and end dates appropriately. Then a new Range was made that held the Result to be added data, and along with the old Range it was added to the children of the new expanded Range.

As can be seen from the last tree produced, the largest start-end date pair encapsulates the smaller start-end date pair, and this is done recursively. This allows for when a list of Results needs to be produced, for example for a Timeline, and the aim is to show which Results happened around the same time period, as Results that overlap with dates are encapsulated.

Result	Start Date	End Date
1	12-12-2016	18-12-2016
2	13-12-2016	20-12-2016

Figure 5.5: Example Results Start and End Date



Figure 5.6: Pre-existing Range Tree



Figure 5.7: Resulting Range Tree after adding the Result: 13-12-2016 → 20-12-2016

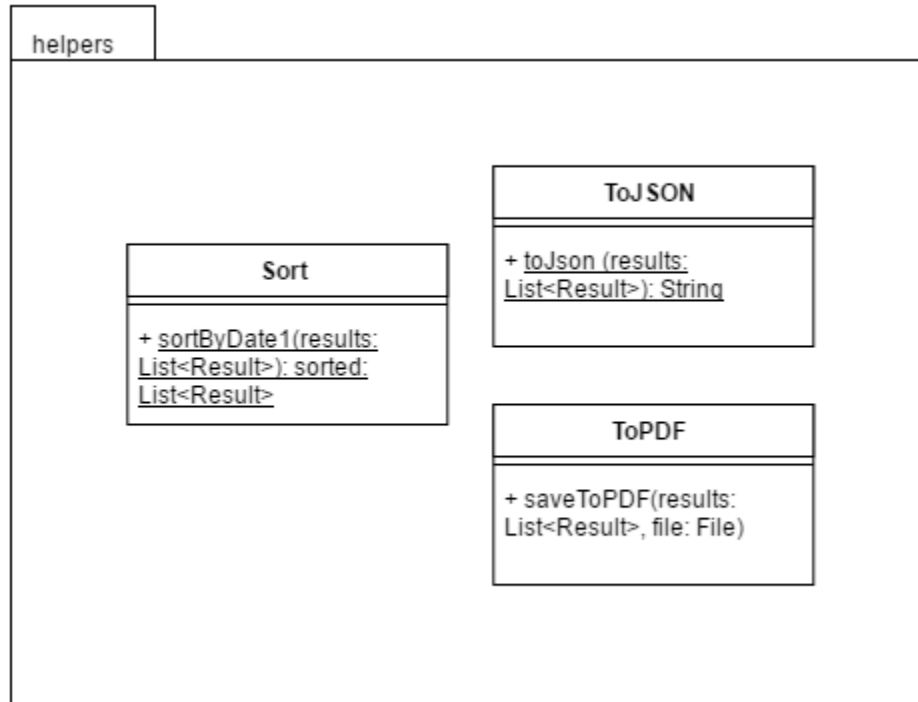


Figure 5.8: Resulting Range Tree after adding the Result: 12-12-2016 → 18-12-2016

#### 5.3.4 Helpers Package

The helper package holds utility components used throughout the system (see Figure 5.9). It provides sorting of lists, and the production of JSON and .PDF files for given list of Results. It is a package used in the refactoring of the logic of the system. Refactoring is done to remove repeated components and operations. Thereby having them in one place only instead of being duplicated around the system. The main advantage is that when a change needs to be made, it is made in one place, but if the operations are duplicated throughout the system then that change needs to be carried out at each place. The common functionality that has been placed in one component to use throughout the system includes sorting Results by their dates or by the number of days between the start and end dates.

Figure 5.9: UML of the Helpers Package



## 5.4 Design Patterns

Design Patterns are general solutions to common problems in software development<sup>3</sup>. The solutions usually include a system architecture to follow. For this project, the main design patterns are Singleton and Observer.

**Singleton** - This design pattern ensures that only a certain number of instances of a component are available<sup>4</sup>. In most cases the number of components is restricted to one. The advantage of this pattern is that it allows for global data used throughout the system to be available in one area. As in the System component the NLP processing tool is held, along with the system state, which is used throughout the system at different places, it is beneficial to have this component follow the Singleton pattern.

**Observer** - This design pattern allows an observee to notify a list of its observers. The observee can be a model, and the the observers its view. This is used to separate the logic of the system from its user interface. It allows for the back-end of the system to be developed completely independently from the front-end. In this system, the front-end will be notified by the back-end. This is then strengthened further through the different system states available.

<sup>3</sup>[https://sourcemaking.com/design\\_patterns](https://sourcemaking.com/design_patterns)

<sup>4</sup><http://www.journaldev.com/1377/java-singleton-design-pattern-best-practices-examples>

As the back-end begins processing documents, its state is changed, the front-end can then use this to modify its view appropriately, by retrieving the relevant data and showing the relevant options for that state. If the system were to be developed further, with other graphical interfaces used or added to the pre-existing UI, it can be added in without any issues. Since the logic of the system is independent of its view.

**Model-View Controller** - It is mentioned as a main design pattern of the system, because it is similar to the Observer design pattern. In that it separates the view of data (a model) through a controller. The controller manipulates the data, and applies the appropriate changes in the view. It separates the logic of the system with its view, like an Observer pattern. This design pattern is enforced by many Graphical User Interface (GUI) frameworks, including the one used in this project.

Other design patterns are available, however these are the most relevant ones to this project. While the solution for the multi-threading is not a design pattern, it is worth mentioning that precautions have to be taken to independently process each document, which includes the previously mentioned semaphores. //singleton, model-view-controller, describe, explain why, and what other options were available

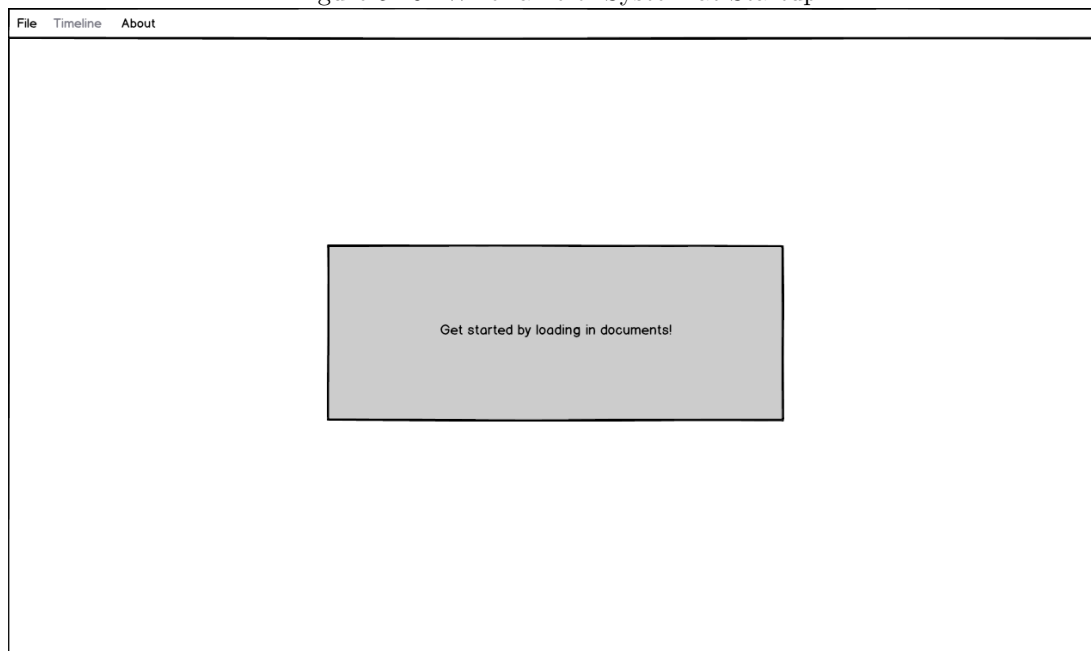
## 5.5 UI

The system is intended to be used by law professionals. However, it can be used by anyone that requires its services of producing timelines, as it will be open-source. These considerations are taken into account during the development of the User-Interface. For example, as the main data representation of the system is the timeline of events, it is clear that it should occupy a majority of the screen. Options that are not relevant to the current state of the system should be unavailable, while options that are most likely to be used should be highlighted. This aims to achieve the visibility objective of the project.

As the main actor can have any level of computer experience, it is important that they are not overloaded with options, and instead are just presented what was is needed. For example, when the system is initially launched, no timeline can be presented as no documents are available to be processed. Thereby, in the Figure presented below (see Figure 5.10) of the launch wireframe, the user is invited to begin using the system by loading documents. It should be noted that a wireframe is a mostly colourless screenshot of what the system should look like.

From this screen the user can load documents and select the reference dates used for these

Figure 5.10: Wireframe of System at Startup



documents. During this period, the back-end of the system can load any resources it requires to process documents. Once the documents have been chosen by the user, they are presented with a loading dialog, to inform them that the system is processing. After this the timeline should be presented (see Figure 5.11).

For each event produced by the system, a timeline row will be produced. With each event an option to edit its data is available, which will provide a dialog (see Figure 5.12) that includes the option to delete an event, and the option to view the document that produced the given event (see Figure 5.13). In addition, the user is provided valuable information such as which documents have been loaded (with the option to remove them and add to them), and the ability to save the timeline. It should be noted, that as can be seen from Figure 5.10 the Timeline option is unavailable as there is no timeline for the user to interact, while in the Figure 5.11 the option is enabled.

In the Document Viewer (see Figure 5.13), it should be noted that the relevant sentence that produced the event, which the user interacted with to view this, is highlighted. This allows the user to see in context the event, as they can read the text surrounding this, giving them a better understanding of what occurred. This also allows the user to swap between the system and the documents, viewing how each event has been produced, and judging how effective the system is. In addition, if the user is interested in a certain event, they can jump to the document and just read the part related to that event, instead of having to read the entire

Figure 5.11: Timeline of System

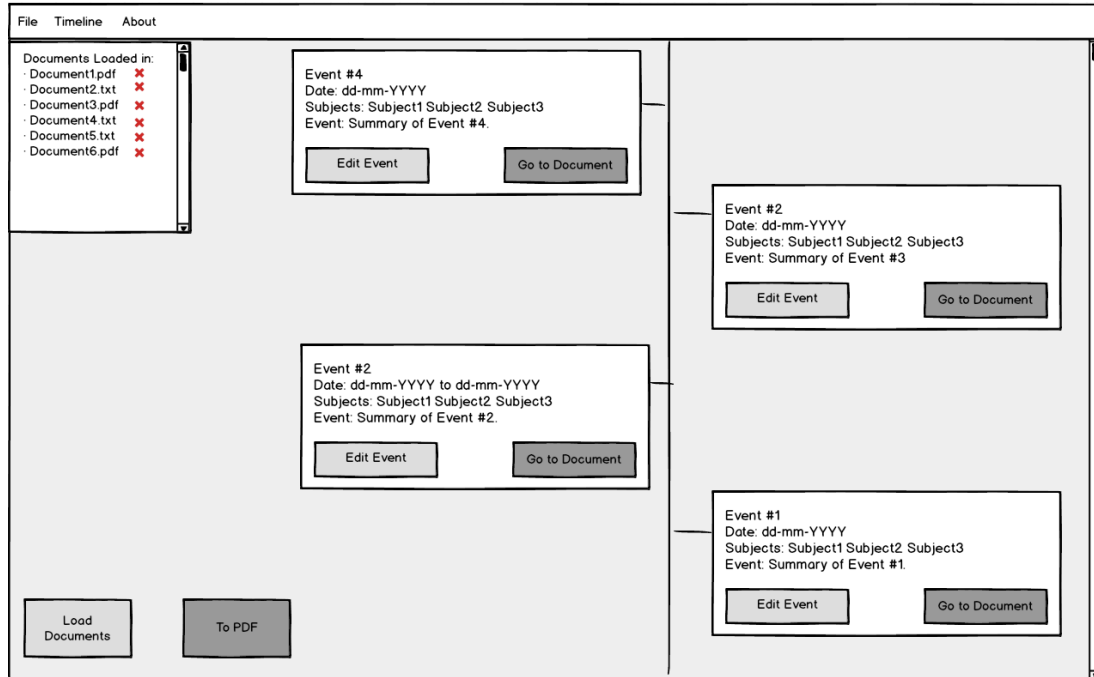
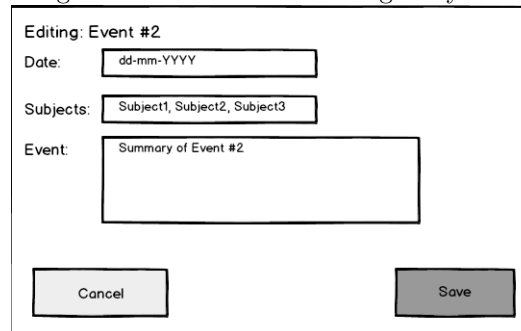
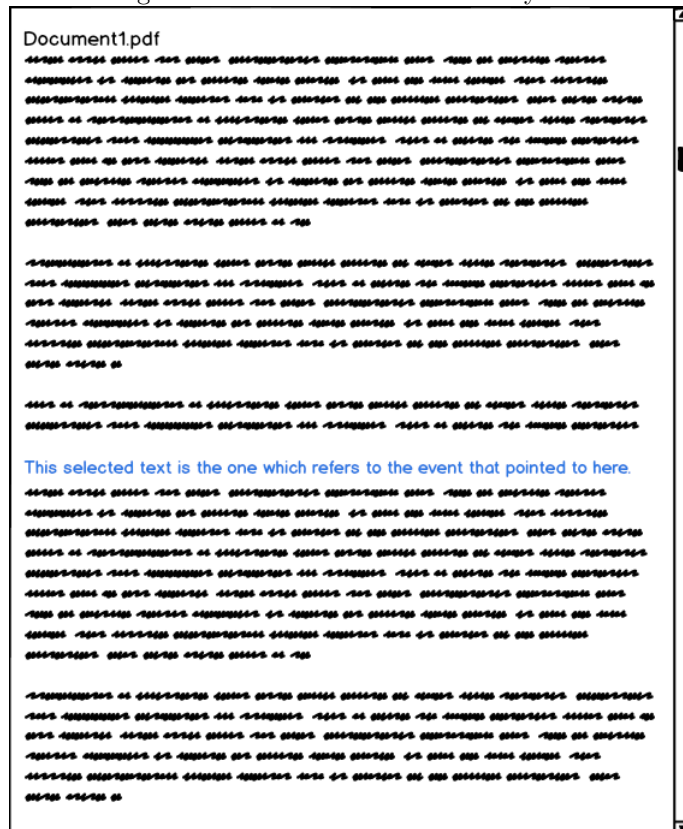


Figure 5.12: Edit Event Dialog of System



document just to obtain more information on one aspect of it. This being extremely beneficial for large documents, as it is both time-efficient, and thereby cost-effective, as the user can focus on that specific aspect of the document (i.e. the text surrounding the sentence that produced the event) instead of having to re-read the chapter, or even the whole document. It should be noted that this tool should not replace the reading of highly sensitive law documents, as the user may require the full context of the document, but it can serve as a time-effective tool after the user has done this. Since it allows them to skip through the document event-by-event, and focus on certain events and areas of the document that they may need to revisit. //wireframes, reason for this, why no color, why no other layout?, update to new system

Figure 5.13: Document Viewer of System





# Chapter 6

## Implementation

-How implemented

### 6.1 Approach

The development approach taken focused on the business-logic, or back-end, of the system. From the two main possible development methodologies, Waterfall and Agile, the latter was used. In Waterfall, the development process is a sequential process, where the development is considered as a sequence of phases that are completed one after the other (cite). In Agile, the focus is on adaptive planning, evolutionary development, and continuous improvement. The advantage of using an Agile approach over a Waterfall approach is that new features can be implemented into the system easier.

The most used way of using Agile methodology is through Sprint cycles. These are short development cycles, where a set of features must be implemented, alongside their tests to ensure the correctness of features (discussed in the Testing section). For this project, the cycles combined with the supervisor meetings. Since in them, the new features that were implemented were discussed alongside the new features that were to be implemented in the next cycle.

A clear example of the advantage of this system was when a new timeline view was suggested. In this new view, rather than having the events row by row, events that occurred during the same time period should be grouped. In addition, events that happened within that time period should be encapsulated by the larger events. This could be implemented in the system, due to the separation of the business logic and the view, and the development approach used. In a Waterfall model, the development is more structured, and thereby it is extremely

useful for static requirements, i.e. requirements that will not change. However, in this case it would have caused issues in implementing the new view as it would require going up the Waterfall if the view of the system had already been implemented, or waiting until that step of the waterfall had been reached.

While the Agile methodology is mostly used in software development teams, it can be applied to single development projects. Since the structure allows for reviews of features which can be matched with supervisor meetings, and changes in the requirements of the project.

## 6.2 Tools & Software Libraries

-development environment -why used that environment -software libraries (include an example use) -why

The development environment of the project is a 64-bit Windows 10 machine, with a Intel Core i7-6700HQ CPU at 2.60GHz and 16.0GB of Random Access Memory (RAM). It includes a Java Intelligent Development Environment (IDE), with Git for version-control, and Gradle for dependency management.

The use of version-control allows development of features separate of working code, and only adding them to the working version if the required tests pass. It should be noted that Git flow was used. This involves having a develop branch with the newest working features of the system. The master branch will only contain the latest fully implemented working version of the product. This allows for mistakes in development to be rolled-back to a state where the system worked correctly.

Gradle allows for libraries to be regarded as dependencies of the project. Such that when the system is ran on a separate machine, it will retrieve all the missing libraries used in the project before compiling and running the program. This allows for the system to be shared to other users, without having to include the libraries with the distribution of the code, as the required libraries and the version will be downloaded to the users system when they run the command:

**gradlew run.**

Where the gradlew is a wrapper for gradle, such that the user does not even have to have Gradle in their system to launch the system. This provides obvious advantages of portability and general use.

As mentioned in the Background chapter, multiple libraries exist to aid the task of Natural Language Processing. These are especially needed for the Named Entity Recognition(NER) and Text Summary task. As an NER annotator will tag certain words, or collection of words

into predefined categories such as People, Companies, Locations, and Money. This is extremely useful for the task of identifying dates in sentences, and the subjects described in the sentences. In addition, these tools can aid in the tagging of words using the P.O.S Treebank, which is required for the implementation of the Hedge-Trimmer algorithm [4] for headline generation (i.e. summary of a sentence). The main advantage of using libraries for this task over developing these annotators, is that building such an annotator requires multiple developers and many years of work. This can be seen from the release history of the StanfordCoreNLP tool which initially released in 2010, but still in October 2016 new releases have been made<sup>1</sup>. The main two NLP tools are Apache's OpenNLP<sup>2</sup> and Stanford's CoreNLP<sup>3</sup>. For this project the Stanford's tool was used in the implementation, as it provides an extended documentation and examples of using their tools, along with specific sections for each of their annotators. The Stanford tool is the main library used throughout the project, as the project is reliant on its NER and POS annotators (cite the stanford annotators). It comes with models, that are loaded during the initialisation of the system. These models are used in the annotators to determine whether certain words fall in predefined categories, or which POS tag should be given to them, through the use of statistics that are based on the models.

As the two main NLP libraries available are Java implementations, the decision was made to build the system in that language. It would be problematic to build the system in a different language to its libraries, as it would require to make the two programming languages communicate with each other, which can cause unpredictable problems in the development and execution of the system.

Additional libraries in the development include JUnit for Unit testing. This allows for features of the system to be developed and for them to be tested for correctness. With the addition of the Git flow, when new features are developed, they are done on a separate branch. Before they are joined to the latest working version of the system the tests for other features and the current developed feature can be ran, thereby ensuring that the system is still working as expected even with the new feature. Unit testing allows for part of a system to be ran, and then to produce a result and compare it to an expected result. The test would then pass if the results match. Testing will be further discussed in one of the following sections.

Libraries for text extraction of .pdf and .docx file types are required, as the encoding of these files is not in a plain text model. Therefore the Apache POI<sup>4</sup> and the Apache PDFBox<sup>5</sup>

---

<sup>1</sup><http://stanfordnlp.github.io/CoreNLP/history.html>

<sup>2</sup><https://opennlp.apache.org/>

<sup>3</sup><http://stanfordnlp.github.io/CoreNLP/index.html>

<sup>4</sup><https://poi.apache.org/>

<sup>5</sup><https://pdfbox.apache.org/>

are used. In addition to text extraction, the PDFBox library along with the Apache Commons library allows the creation of PDFs (with text wrapping), which is required to save the timeline as a PDF. The Google GSON<sup>6</sup> library is used for the creation of JSONs, which is required to produce an intermediate JSON of the timeline. The RichTextFX<sup>7</sup> library along with JavaFX are used to build the graphical interface of the system. All of the libraries included are provided with licenses that allow its use in systems, along as the system is made publicly available, which will be done as the resulting system will be open-source.

## 6.3 Issues

Two main issues arised during the implementation of the system. The creation of exact dates for named entity dates and the creation of a encapsulated timeline. In addition, a minor issue in the system is input of documents that are grammatically incorrect.

### 6.3.1 Named Entity Recognition (NER) of Dates

The StanfordCoreNLP tool, allows the resolution of temporal expressions. To explain this, an example is presented. For example, the Stanford tool allows for reference dates be used when a document is processed. When the tool tags a temporal expression as DATE, it allows for this temporal expression to be normalized. The annotator treats each word in the sentence as a mention. To identify its named entity recognition tag, the following is done on the mention:

**`mention.get(CoreAnnotations.NamedEntityTagAnnotation.class).`**

If it is a temporal expression that was tagged, the result of the operation is a String DATE (to identify it as a date). Thus, from the mention, it can be normalized using:

**`mention.get(CoreAnnotations.NormalizedNamedEntityTagAnnotation.class).`**

The Stanford tool will attempt to produce a date in the ISO 8601<sup>8</sup> format. As can be seen from the format, it can produce exact dates of the type dd-MM-yyyy. Where dd is an integer value from 1 to 31, MM is the month as an integer value from 1 to 12, and yyyy the year. In addition, if BC dates are used, at the start of the normalized result a '-' is added. Using the ISO standard, a method was written to process these dates. However, in addition to the possible dates given by the ISO standard, Stanford builds on top of the standard, producing 3 additional Date normalizations.

---

<sup>6</sup><https://github.com/google/gson>

<sup>7</sup><https://github.com/TomasMikula/RichTextFX>

<sup>8</sup><https://www.cl.cam.ac.uk/~mgk25/iso-time.html>

The normalizations refer to temporal expressions that are ambiguous even with a reference point. For example, the temporal expression "now" would produce a normalized NER: "PRESENT\_REF", i.e. a reference to the present moment. For a temporal expression that refers to the past, e.g. "...they once used to...", "once" would be normalized to "PAST\_REF", i.e. a reference to the past of this moment. For a temporal expression that refers to the future, e.g. "In the future...", "future" would be normalized to "FUTURE\_REF", i.e. a reference to the future after this moment. The issue with these normalizations is that they do not allow for the comparison of events used to sort them, as the start and end dates of events cannot be compared. To allow for comparison, the most possible general dates for these references are derived. Since a "PRESENT\_REF" refers to the present moment, it can be deduced that it represents the moment in which the text was written in, as that is the time context in which the author wrote it in. Thereby, the decision was made to produce as a general date for "PRESENT\_REF" the reference point provided by the user. Since the reference point is supposed to be the date in which the text was written in, it would be appropriate to link a reference of the present moment to it. The user can change the reference point to when-ever they would like not just the assumed date of the written document, but a present reference should match the base date used by the system to determine all other ambiguous dates. For "PAST\_REF" and "FUTURE\_REF" a range of dates is used, i.e. a start and end date. For the former, the start date of the era is used, i.e. 01-01-0001, and an end date to the reference point. Since it can be determined that an ambiguous mention of the past would fall anywhere within that time period, however an exact determination cannot be made as the temporal expression is not precise enough. For the latter, the start date is the present moment, i.e. the base date, and the end date is the last possible allowable date in the system, i.e. 31-12-9999. Since it would be appropriate for a mention of the future would refer to a moment from now (i.e. the present moment in which the text is presumed to be written in), up until the end of times. However as a limitation to our system, the end of times is considered the date 31-12-9999. This detection part of the normalized NER for Dates is presented as a snippet of the method that produces the exact dates from Normalized NER dates (see Figure 6.1).

Even though the StanfordCoreNLP library is well documented, it was difficult to find all the template outputs of the tool when Normalizing NER Dates, as no file was initially found that pointed to all the outputs (specifically the "REF" outputs). However, afterwards it was found that the tool uses TimeML for the normalized dates, thus the templates could be found

9.

---

<sup>9</sup><http://www.timeml.org/timeMLdocs/TimeML.xsd>

```

private ArrayList<Date> getDate(String date) {
    ...
    /* Set up variables for processing */
    if (onlyPastRefPattern.matcher(date).matches()) {
        //past so make range from 0001-01-01 -> base date (range)
        if (yearMonthDayPattern.matcher(baseDate).matches()) {
            //base date has the format yyyy-MM-dd
            /*split it and set the values in date1,month1, year1 of
            the start year 01-01-0001*/
            /*and the end date values of date2, month2, year2
            to the base date data*/
            ...
        }
    } else if (onlyPresentRefPattern.matcher(date).matches()) {
        if (yearMonthDayPattern.matcher(baseDate).matches()) {
            //base date has the format yyyy-MM-dd
            /*set day1,month1,year1 to the data in the base
            date data*/
            ...
        }
    } else if (onlyFutureRefPattern.matcher(date).matches()) {
        if (yearMonthDayPattern.matcher(baseDate).matches()) {
            //set the day1,month1,year1 to the base date data
        }
        //set year2,month2,day2 to the end year 31-12-9999
    }
    ...
    /* date1,date2,month1,month2,year1,year2 are
    then used appropriately to generate dates used
    for the event that holds this Timeline Date */
}

```

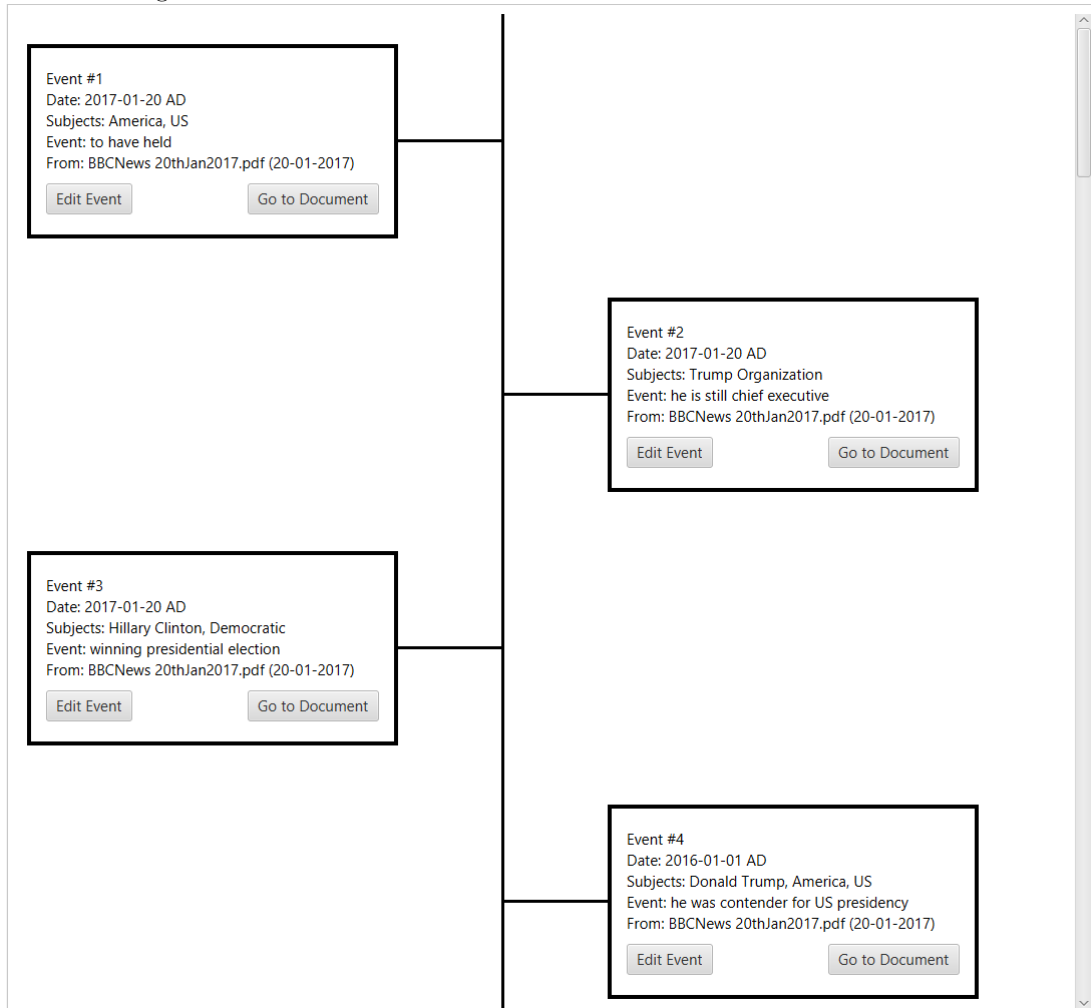
Figure 6.1: Part of the Implementation of Resolution of Normalized NER Dates

### 6.3.2 Encapsulated Timeline View

The initial representation of the events was a traditional timeline (see Figure 6.2). This view is effective when the events are on separate time periods, as it presents them one after the other in a sequence. However, when there are multiple events that occur during the same time period, they still appear one after the other. The issue is that unless the user specifically looks at the dates associated to the event, it will look at a first glance as events occurring in different time periods. This clearly violates the visibility objective of the user-interface of the system. A solution to this issue is to provide two views, the traditional timeline view which is effective at displaying events that have disjoint dates. The other view, is one where dates encapsulate other dates, and hold the events that occur in that time period. For example, if there is more than one event that occurs on the "25-01-2017", then instead of listing them both of these events are below the same date. This can cause a "bin-placing" problem. This is where there are a set of bins, or in this case a set of graphical events, and they need to fit in a finite area (cite), in this case a box of certain width and height. However, this problem can be avoided through the use of scrollbars, which provides a container of infinite height, thereby all the bins (graphical representation of events) can be placed. This view has been called the "Range View". However, producing this view, requires placing the Results (the events of a set of documents) in Ranges (a data structure that can have one date, or a start and end date). The theory was discussed in the Design Chapter.



Figure 6.2: Screenshot of the Traditional View of the Timeline of Events



The theory behind the Range View, is having a list of Ranges, where each Range is a root node of a Tree of Ranges. Where each Range may hold zero or more Results (i.e. events), and a set of children Ranges (zero or more). When the timeline needs to be produced, the list of Roots is iterated over, assuming the list of Roots has been sorted in the order of the start date, for each a GridPane is made. A GridPane is a layout which consists of rows and columns that can contain subviews (where a subview is a view, i.e. a graphical component). In the first column of the of the layout, the current Range's data is placed (i.e. the date(s) and the Results held), and in the second column the layouts of the child Ranges is recursively made. The algorithm is presented in Figure 5, it is done for each root Range in the list of Ranges to be created.

---

**Algorithm 5:** Pseudo-Code of the Recursive Production of the Range Layout

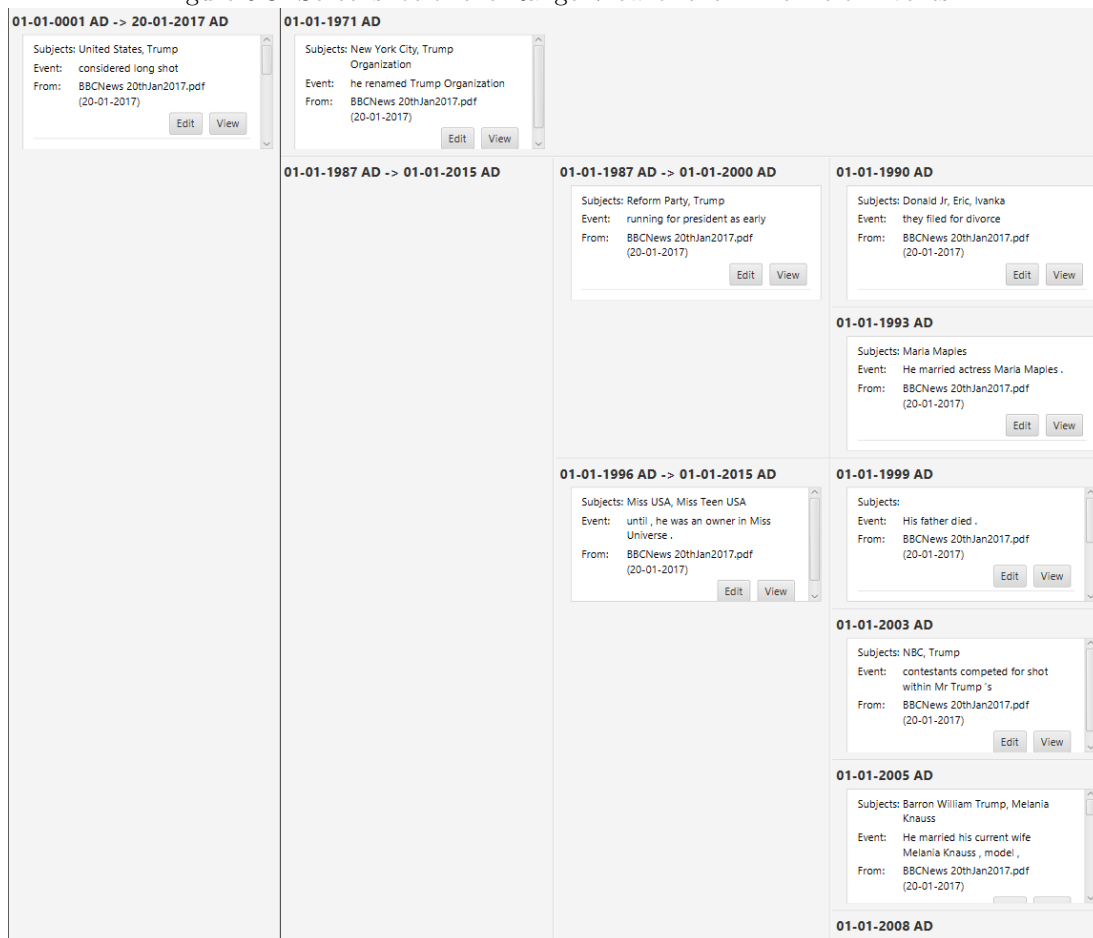
---

```
1 function getRangeLayout(list l);  
   Input  : A list of Ranges, of size  $n$ , to add in the first column  
   Output: a layout that encapsulates the Ranges passed in the input, and their child  
           Ranges  
2 GridPane toReturn;  
3 toReturn set the number of rows to the size of the input;  
4 toReturn set the number of columns := 2;  
5 for  $i := 0 \rightarrow n$  do  
6   Range := input list at  $i$ ;  
7   set up layout for this Range, and set it in toReturn at position  $(i, 0)$ ;  
8   set its column span at  $(i, 0)$  to remaining;  
9   get layout for the children of this Range := getRangeLayout(range.children);  
10  set this layout in toReturn at position  $(i, 1)$ ;  
11 end  
12 return toReturn;
```

---

The full implementation can be found in the Appendix. In the implementation, it was decided to include 3 columns, to have a separator between a Range and its children Ranges, to improve the visibility of the system. Allowing the user to differentiate a Range and its child Ranges, and thereby differentiate the Results they hold. The individual layouts of Range is the listing of the Results in that Range (i.e. in the time period of that Range), and the date or start and end date for that Range. An example look of the Range View can be found in Figure 6.3.

Figure 6.3: Screenshot of the Range View of the Timeline of Events



### 6.3.3 Incorrect Input Documents

An important issue relating to the input documents is their format. NLP tools such as StanfordCoreNLP require a certain constraints to be assumed onto the input documents to be able to properly process them. Since these tools require assumptions to apply certain grammatical rules, it is extremely important that the documents are in correct English grammar. This was discussed in the Background Chapter along with a discussion of NLP tools being applied to Tweets, and how badly they performed. It is important that the documents are in English, as the models loaded by the StanfordCoreNLP in this implmentation are the English ones. All the other languages supported by the tool<sup>10</sup>, such as German, Spanish, and even Chinese require other models. The main implementation focus for the tool was the English language, but from when it was developed it was considered to be extended to other languages.

An issue that was discovered, was that separate sentences that a seperated by a period,

<sup>10</sup><http://stanfordnlp.github.io/CoreNLP/human-languages.html>

but not a space following it, are treated as one sentence. The system still continues to work, however it can cause that only half of the events are detected, if every two sentences that have events are separated just by the period (and not the space). In addition, the summary would be applied to second sentence, as the rule of the algorithm details that the lowest-leftmost "S" subtree (i.e. sub-sentence) is picked. The start and end date of such an event, would therefore be the lowest date of the two events, and the highest date of the two events, due to how the dates of events are encapsulated in a TimelineDate object (that parses Normalized NER Dates, and updates the start and end dates it holds if a new min or max is found).

This issue can not be prevented, as it would require manipulation of the input documents, and it can be very likely the case that the user does not want the system to manipulate the input but rather just process it. Hence, it will be advised to users to ensure the documents are in English and grammatically correct. As the system will be used by law professionals, that are handling law documents, it can be assumed that these documents would follow this format, as law documents tend to be very formal texts. -ner dates (explain, then present how to solve it through examples of code) -new timeline view (present how to solve it through examples of code) -minor issue of incorrect text

## 6.4 Testing

The focus of the testing in the system were Unit Tests. A Unit Test is when individual units (or pars) of source code of a system are tested to determine whether they are working correctly (cite). As the system was implemented in Java, the library used to aid this is JUnit<sup>11</sup>. Unit tests are primarily done on the back-end, or logic, of a system, as they focus on these parts to work correctly, and not the interaction of a user with the system. To test a users interaction with a system, Instrumentation tests are carried out. These involves emulating the users interaction with the system. Both the Unit and Instrumentation tests can be automated, such that they are carried out one after the other without the involvement of the developer.

The reason as to why only Unit Tests were used, is that the systems primary focus is its processing of documents, and not the graphical representation. The representation is used to display the results, however as disussed previously, the system could also just be used to process the texts, and the intermediate JSON used for a third-party visual representation. Hence, Instrumentation tests were not carried out.

A total of 35 tests were developed. The main advantage of these is that when new features

---

<sup>11</sup><http://junit.org/junit4/>

are developed, to ensure that the system is still functioning correctly, the tests can be ran. If the test cases are appropriate, then it can be assured that the system will work correctly with the new features. This is aided additionally through the use of Git, where the features are developed on a separate branch of the current working version of code, and are only merged into the working version if all the test cases pass.

The tests focus was on the Engine (the component that takes as input text and produces lists of Results), the processing of Files (test files were used in this case), the production of Ranges, the changing of the systems states throughout the processing task, the parsing of Normalized NER dates, and the production of JSON's from a list of Results. Tests worked through assertions. Assertions are made where the expected output must match the actual output.

The tests were divided into three categories, a simple test, an intermediate test, and a complex test where all possibilities of input are tested. For example, for the production of Ranges, the complex test case expects multiple Range trees of different heights to be produced. The tests can be found in the Appendix. Due to the benefit of Gradle, the tests can be ran using the command:

**gradlew test**

or, in the case the libraries need to be loaded:

**gradlew build).**

In the build command, Gradle will not only run the test cases, but also produce the executable JAR which can be used to distribute the system as an executable to its non-developer users.

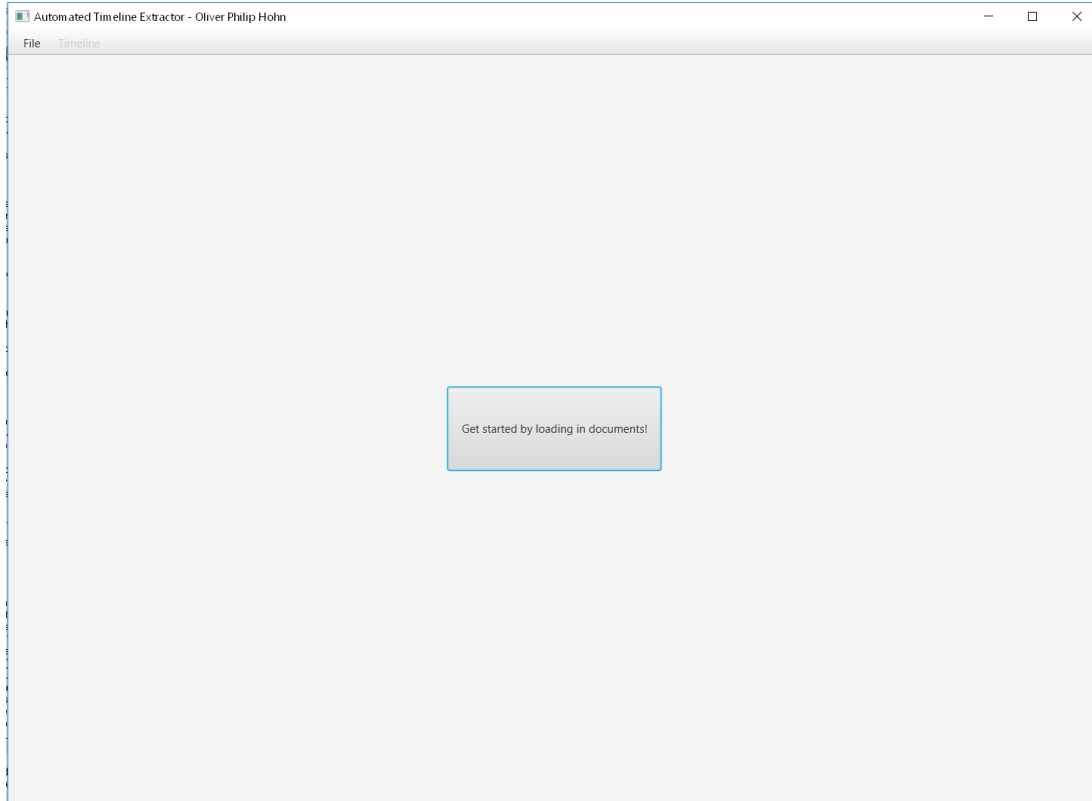
## 6.5 UI

//implementation ofthe UI.

From the wireframes presented in the Design Chapter, the actual User Interfaces (UI) were developed. The screenshots of the different windows are presented in the Figures below (see Figures 6.4, 6.2, 6.3, 6.5, and 6.6). As can be noted, the interface provides the requested functionality through the buttons and menus, but are missing color. Since the focus was on the processing of the text, a color palette was not developed for the system as it is believed that the system will be used for work related tasks, thereby its focus is not on enjoyment but rather functionality, visibility and simplicity. These objectives were attempted to be reached with this UI implementation.

It should be noted that in the Range view (see Figure ??) is zoomable. This allows the user

Figure 6.4: Screenshot of the Start Up View

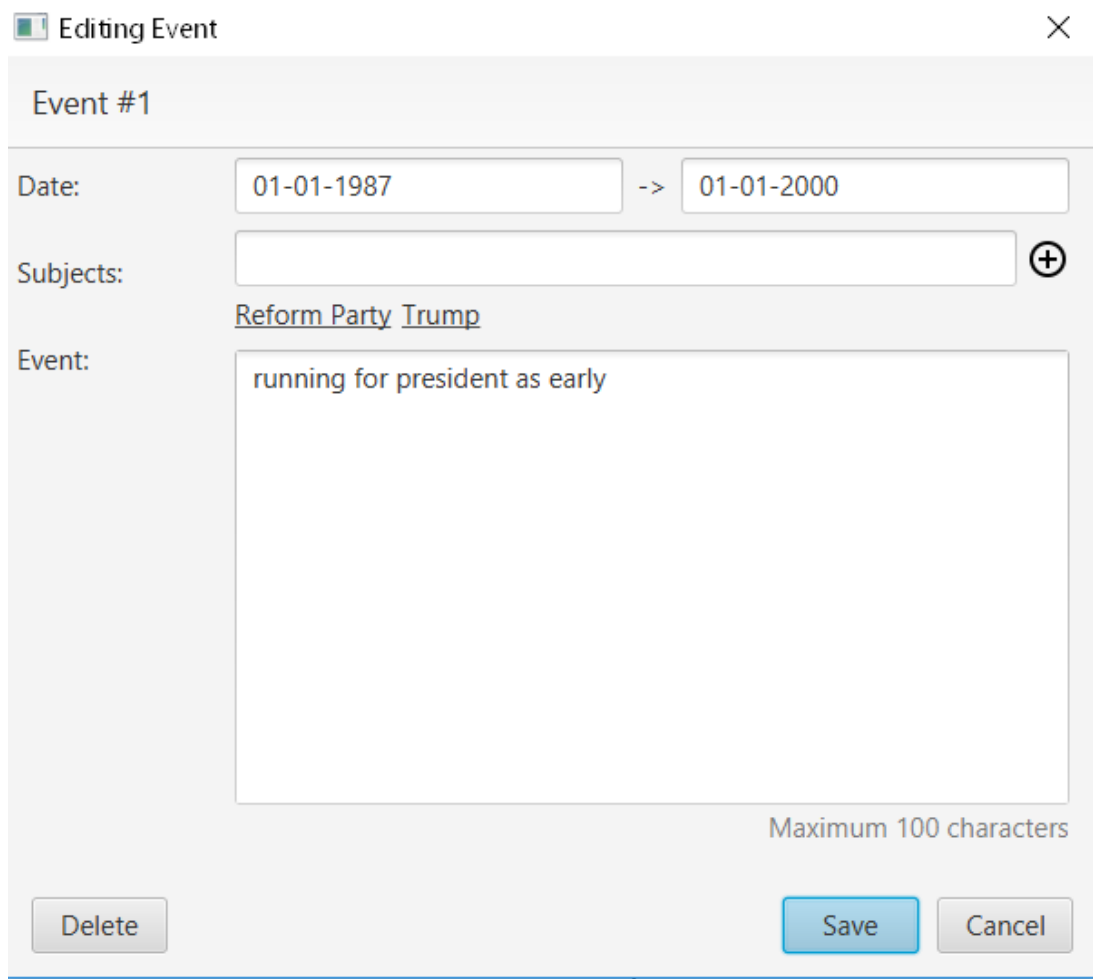


to have a broader image of the Range View, if it is required. It may allow the user to obtain a general picture of what occurred.

The data presented of events includes the subjects and its summary. Since the events are encapsulated in Ranges, it is sufficient to show the date(s) for this Range at the top. Thereby intuitively demonstrating to the user that the following events occurred during that time period. This separates meta data from the event, e.g. when the event occurred, from the actual data which is what occurred in the event and what are the subjects of interests of the event. In addition, each event is accompanied by two buttons, which are both the relevant main operations: to edit the event (which includes deleting it, as can be seen from Figure 6.5, and to view the document that produced it.

An advantage of the Traditional View over the Range view, is memory efficiency. As the Range view has the function of being zoomable, i.e. the user can zoom in/out, due to language specific constraints, it cannot be implemented through a ListView. A ListView being a layout data structure where objects of a list are placed row by row. To avoid stack overflows if the list is too large, the ListView will only produce the graphical layout for the rows when they are needed (i.e. they need to be shown), and otherwise delete them. For example, for a list of

Figure 6.5: Screenshot of the Edit Dialog View



The screenshot shows a dialog box titled "Editing Event" with a close button (X) in the top right corner. The dialog is divided into several sections:

- Event #1**: A header section.
- Date:** Two text input fields. The first contains "01-01-1987" and the second contains "01-01-2000", separated by a right-pointing arrow "->".
- Subjects:** A text input field containing "Reform Party Trump". To the right of the field is a circular button with a plus sign (+).
- Event:** A large text area containing the text "running for president as early".

At the bottom right of the dialog, below the text area, is the text "Maximum 100 characters". At the bottom of the dialog are three buttons: "Delete" on the left, "Save" in the center (highlighted in blue), and "Cancel" on the right.

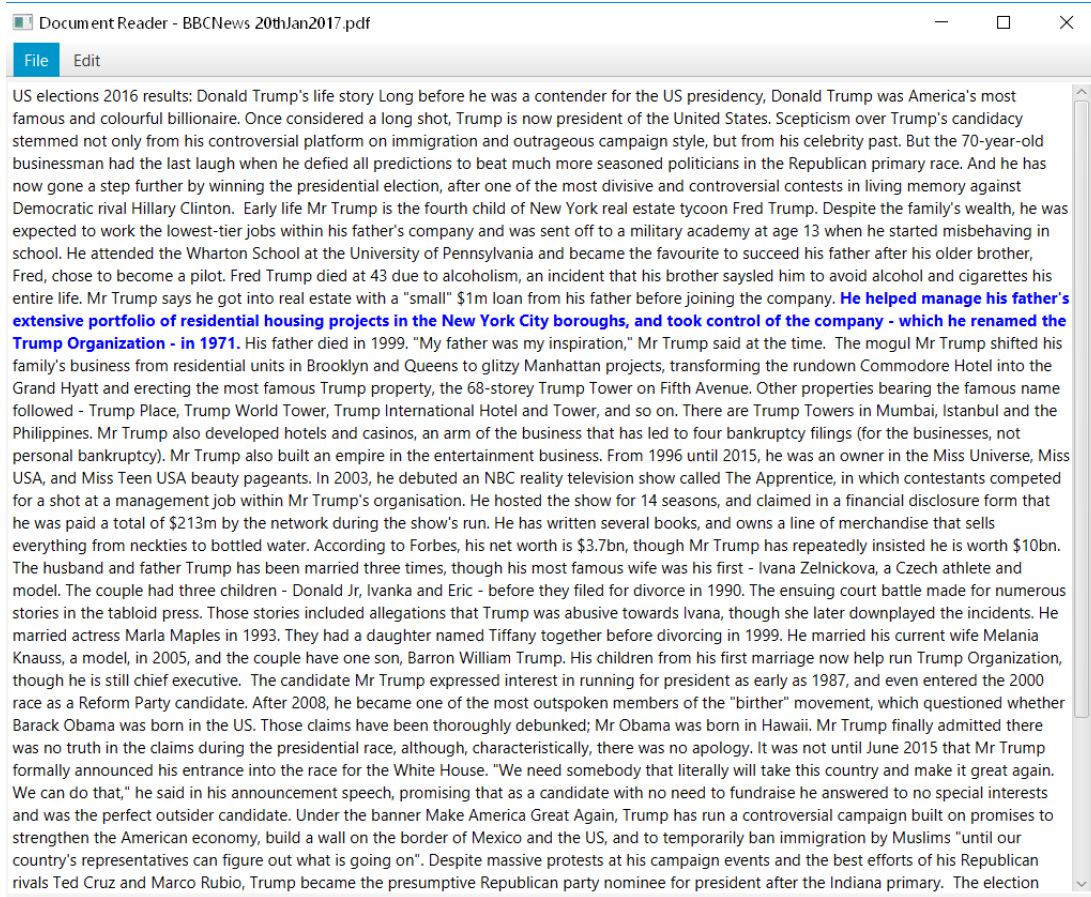
10000 items, it is very expensive to hold in memory the individual layouts of 10000 rows, even when only 5 are being shown. Instead only the 5 that are being shown are held in memory, and the rest are generated as necessary. This is not the case with the Range View, as a tradeoff was made to allow the zoom function. As the memory capacity of personal computing systems (where the tool is intended to be used, but is not limited to) is large, this would not be an issue for the processing of tens or hundreds of events, however for larger sets this would cause memory issues. Thus the traditional view is suggested in such case.

## 6.6 Important Algorithms

//highlight algorithm implementation: processing of files through semaphores, adding to ranges, pdf save and json save

This sections intended use is to present noteworthy algorithms. In specific their implemen-

Figure 6.6: Screenshot of the Document Viewer



tation.

### 6.6.1 Processing Files

The pseudo-code of the processing files was presented in the Design Chapter. It was also mentioned the use of semaphores, where they enforce that only  $n$  processes can acquire their lock, with the  $n + 1$  process having to wait until a process releases its lock. However, the implementation is not trivial (see Figure 6.7). The system must allow a certain maximum number of threads to be ran in parallel to process documents. It must not fall in deadlock and starvation, that is when processes are waiting indefinitely to run, and thereby the system does not move ahead. Thus can occur when semaphores are used, and they are not released when a process finishes its task, in this case processing a file and producing a list of Results. Thus a callback is used when the Threads are used, this callback is to be used when the Thread finished it task. It will ensure that the semaphores are released. The processing of documents ensures that when an error occurs, a result is still produced, thus the thread should not finish



and not use the callback. However, necessary multi-threading precautions must be considered. When the callback is used, data is also transferred (specifically the Results of processing the document) which can cause concurrency issues when two threads attempt to add to the result list at the same time. Fortunately, Java provides the **synchronized** keyword for methods (see Figure 6.8). This ensures that no two threads may run the method at the same time. One thread must finish its execution of the method before another one can begin its execution. Thus dealing with the issue of adding to a list at the same time.

Threading precautions have been taken throughout the project. Where threads and data are used, it is ensured that the data can be processed independently of itself to allow for parallelism. Where this is not the case, synchronization and semaphores are used to ensure that concurrency issues are dealt with.

### 6.6.2 Building Ranges

The algorithm for building Ranges out of Results was presented in the Design Chapter. However, it was not described how it can be checked whether or not a Result can be added to an existing Range. When a Result is attempted to be added to a pre-existing Range, a **add()** (see Figure 6.9) method is called. In this method, it is initially checked whether the Result should be attempted to be added to this Range. This check involves looking at the dates of the Result and checking whether they are fully or partially encapsulated by the Range's dates. If this is not the case, then the attempt of adding the Result fails, otherwise the algorithm continues. Next it will be checked whether it can be added to any of the nodes of the Range tree, if there is any node that holds the exact same date(s) as the Result, i.e. the Result should be held by this Range. If this is the case then the Result will be added and that node, however if this is not the case, then it can be determined that the Result belongs to this Range, however its position in the tree needs to be created or a Range that partially encapsulates it needs to be expanded. This involves traversing the tree and finding the position where the new Range note should be placed, or expanding a Range and moving subtrees to maintain the structure of the Range tree.

### 6.6.3 Saving Results

Saving the results of the processed documents, is divided into two parts: saving them as a PDF, or as a JSON.

In the PDF system, the file that is saved, is a graphical representation of the events, using

```

public List<Result> processFiles(List<File> files , List<FileData> fileDatas) {
    //semaphoreFinished initially value of 0
    maxNoOfThreads =
        BackEndSystem.getInstance().getSettings().getMaxNoOfThreads();
    //set the Max number of Threads that can run in parallel
    semaphore = new Semaphore(maxNoOfThreads);
    //should only run if we are not Processing
    if (BackEndSystem.getInstance().getSystemState() !=
        SystemState.PROCESSING && files.size() == fileDatas.size()) {
        filesToGo = files.size();
        //are now processing
        BackEndSystem.getInstance()
            .setSystemState(SystemState.PROCESSING);
        for (int i = 0; i < files.size(); i++) { //for all the files to process
            File file = files.get(i);
            FileData fileData = fileDatas.get(i); //should be the same
            try {
                //will wait if there is already maxnoofthreads running,
                //until one releases the semaphore
                semaphore.acquire();
                //process file
                Thread thread =
                    new ProcessFile(file , ProcessFileCallback.this , fileData);
                //pass a callback , that can be called when file is finished
                //adds generated Results to the result list
                thread.start(); //start processing the file
            } catch (InterruptedException e) {}
        }
        try {
            semaphoreFinished.acquire();
        } catch (InterruptedException e) {}
        return results; //could run, so return the results
    }
    return null;
}

```

Figure 6.7: Implementation of Processing of Files

the traditional timeline view. It is aided through the use of the Apache PDFBox and Commons library. The libraries work like a painting tool. The pages are canvases, where lines and text can be drawn on at specific positions. These positions are given by coordinates  $(x, y)$ , where the top left of the page is the origin, i.e.  $(0, 0)$ , and the x values increment towards the right of the page, and the y values towards the bottom of the page. To build a dynamic system, that can create a pdf for any number of Results, it requires certain constraints. Such as the maximum number of events to be displayed on each page, and the size of the text of the summaries and subjects. Due to issues with the tool, it is not possible to continue on the next page of the PDF, hence it is limited to how many events are shown and the size of the text (as it can overflow to the next page). However, since the text that is used in the summary, should be short (since

```

public synchronized void callBack(ArrayList<Result> results, FileData fileData) {
    //we finished processing a file
    filesToGo--; //one less to look at
    //add the results to the list held
    this.results.addAll(results);
    //release semaphore
    semaphore.release();
    //check if we have processed everything,
    //if so release the finished semaphore
    if (filesToGo == 0) {
        //has processed
        BackendSystem.getInstance().setSystemState(SystemState.PROCESSED);
        //has returned the results so we finished
        BackendSystem.getInstance().setSystemState(SystemState.FINISHED);
        semaphoreFinished.release();
        //value is now 1, so the thread that was acquiring can continue
    }
}

```

Figure 6.8: Implementation of Callback after Files have been processed

it is a summary), the constraint can be applied. Therefore, it can be assumed that there is a max size of the container that holds one event. This allows for every container to be of this max size, to avoid the issue of resizing containers depending on the amount of text. Therefore, the system can be considered as drawing containers on the page and filling them with the data, moving along to the next position and repeat. Hence the task, in the implementation, has been broken down to padding, writing text, and drawing the rectangle. Since in the traditional view, the layout of the events are one on the left and the next on the right, the tasks have to be changed minimally to allow for this. The full implementation can be found in the Appendix. As an example, the Figure 6.10 has been provided to demonstrate the task of drawing for an event that is to be placed on the right of the page.

In the JSON implementation, it is aided by the use of the Google GSON library. The advantage of this library is that it allows for a flexible creation of JSONs. In specific, it allows the developer to specify how an object of a given type should be serialized. In this implementation, the objects to be serialized are Result objects. The data to be added for them is the dates, the subjects, the summary and the relevant data of the file that produced it. This can be done through defining the adapter to be used for the objects of the Result class. In the adapter the resulting **JsonObject** can be created, the data can be set in it, and then return it. For a list of Results, this will be carried out for each Result, and thereby a resulting list of JsonObjects will be returned, i.e. a JSONArray. This can then be saved by the user in a file, and can be interpreted by any third-party system, as this is the use of JSON and the format of

```

public boolean add(Result result) {
    TimelineDate timelineDate = result.getTimelineDate();
    //check constraints
    if (!shouldAdd(timelineDate)) {
        //check constraints if we can even add to this range
        return false;
    }
    //attempt to add through an existing range
    Range toAdd = checkCanAdd(result);
    if (toAdd != null) {
        //add to the results of the given range
        toAdd.results.add(result);
        return true;
    }
    //now we try to extend the range
    return createRangeAndAdd(result);
}

```

Figure 6.9: Implementation of Adding Results to a Range

each event (Result) is clearly defined with key-value pairs. The format of a JSON of a Result is given in Figure 6.11. Where G in the dates is the ERA (i.e. BC or AD) of the date. -main algorithms: for building events, building ranges, pdf save, json save, processing files

```

private void drawOddEvent(Result result ,
    PDPageContentStream contentStream , int position)
    throws IOException {
    //initially y is the top right where this needs to
    //be shown, x starts from the middle
    currentY -= padding; //add some padding to y
    currentX = (int) widthOfPage / 2;
    contentStream.moveTo(currentX , currentY);
    //write the text for the Event
    int lengthOfHorLine = (int) ((widthOfPage / 2)
        - (padding + widthOfRectangle));
    currentX += lengthOfHorLine;
    writeText(result , contentStream , currentX , position);
    //draw the rectangle to surround the text
    drawRectangle(contentStream , currentX , currentY
        - heightOfRectangle);
    //draw the horizontal line connecting event timeline
    currentY -= heightOfRectangle / 2;
    contentStream.moveTo(currentX , currentY);
    contentStream.lineTo(widthOfPage / 2 , currentY);
    contentStream.stroke();
    currentY -= (heightOfRectangle / 2) + padding;
}

```

Figure 6.10: Implementation of drawing for a Result in the PDF timeline

```

{
  "date1":dd-MM-yyyy G,
  "date2":dd-MM-yyyy G,
  "subjects":[] ,
  "event":String ,
  "from":{
    "filename":String ,
    "baseDate":dd-MM-yyyy
  }
}

```

Figure 6.11: Structure of Result (event) JSON

## Chapter 7

# Professional and Ethical Issues

Either in a separate section or throughout the report demonstrate that you are aware of the **Code of Conduct & Code of Good Practice** issued by the British Computer Society and have applied their principles, where appropriate, as you carried out your project. -how dealt with ethical approval? (newspapers used, etc.) -in analysis kept testers anonymous

### 7.1 Section Heading

## Chapter 8

# Results/Evaluation

-present how did analysis, why?, other options (relate to work) -results of analysis -conclusion

### 8.1 Software Testing

### 8.2 Section Heading

## Chapter 9

# Conclusion and Future Work

The project's conclusions should list the key things that have been learnt as a consequence of engaging in your project work. For example, "The use of overloading in C++ provides a very elegant mechanism for transparent parallelisation of sequential programs", or "The overheads of linear-time n-body algorithms makes them computationally less efficient than  $O(n \log n)$  algorithms for systems with less than 100000 particles". Avoid tedious personal reflections like "I learned a lot about C++ programming...", or "Simulating colliding galaxies can be real fun...". It is common to finish the report by listing ways in which the project can be taken further. This might, for example, be a plan for turning a piece of software or hardware into a marketable product, or a set of ideas for possibly turning your project into an MPhil or PhD.

-what have you learned? -how can the project be carried further (neural net for summary, building on the StanfordCoreNLP for detas depending on others)



# References

- [1] Sentence subjects. <http://grammar.ccc.commnet.edu/GRAMMAR/subjects.htm>. URL <http://grammar.ccc.commnet.edu/GRAMMAR/subjects.htm>. Accessed 4 Dec. 2016.
- [2] S. Chopra, M. Auli, and A. Rush. Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of NAACL*, 2016. URL [http://nlp.seas.harvard.edu/papers/naacl16\\_summary.pdf](http://nlp.seas.harvard.edu/papers/naacl16_summary.pdf).
- [3] H. Daumé III and D. Marcu. Noisy-channel model for document compression. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 449–456, 2002. URL <http://www.aclweb.org/anthology/P02-1057>.
- [4] B. Dorr, D. Zajic, and R. Schwartz. Hedge trimmer: A parse-and-trim approach to headline generation. In *Proceedings of the HLT-NAACL 03 on Text summarization Workshop- Volume 5 (ACL)*, pages 1–8, 2003.
- [5] K. Knight and D. Marcu. Statistics-based summarization – step one: Sentence compression. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 703–710, 2000. URL <http://www.aaai.org/Papers/AAAI/2000/AAAI00-108.pdf>.
- [6] D. McClosky and C. Manning. Learning constraints for consistent timeline extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 873–883, 2012. URL <http://nlp.stanford.edu/pubs/dmcc-emnlp-2012.pdf>.
- [7] A. Ritter, S. Clark, Mausam, and Etzioni. Named entity recognition in tweets: An experimental study. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1524–1534, 2011. URL <https://aclweb.org/anthology/D/D11/D11-1141.pdf>.

- [8] A. Rush, S. Chopra, and J. Weston. A neural attention model for sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, 2015. URL <http://www.aclweb.org/anthology/D15-1044>.

## Appendix A

# Extra Information

### A.1 Tables, proofs, graphs, test cases, ...

The appendices contain information that is peripheral to the main body of the report. Information typically included in the Appendix are things like tables, proofs, graphs, test cases or any other material that would break up the theme of the text if it appeared in the body of the report. It is necessary to include your source code listings in an appendix that is separate from the body of your written report (see the information on Program Listings below).

## Appendix B

# User Guide

### B.1 Instructions

You must provide an adequate user guide for your software. The guide should provide easily understood instructions on how to use your software. A particularly useful approach is to treat the user guide as a walk-through of a typical session, or set of sessions, which collectively display all of the features of your package. Technical details of how the package works are rarely required. Keep the guide concise and simple. The extensive use of diagrams, illustrating the package in action, can often be particularly helpful. The user guide is sometimes included as a chapter in the main body of the report, but is often better included in an appendix to the main report. -how to set up (commands) -how to use given pieces of sample text -images

# Appendix C

## Source Code

### C.1 Instructions

Complete source code listings must be submitted as an appendix to the report. The project source codes are usually spread out over several files/units. You should try to help the reader to navigate through your source code by providing a “table of contents” (titles of these files/units and one line descriptions). The first page of the program listings folder must contain the following statement certifying the work as your own: “I verify that I am the sole author of the programs contained in this folder, except where explicitly stated to the contrary”. Your (typed) signature and the date should follow this statement.

All work on programs must stop once the code is submitted to KEATS. You are required to keep safely several copies of this version of the program and you must use one of these copies in the project examination. Your examiners may ask to see the last-modified dates of your program files, and may ask you to demonstrate that the program files you use in the project examination are identical to the program files you have uploaded to KEATS. Any attempt to demonstrate code that is not included in your submitted source listings is an attempt to cheat; any such attempt will be reported to the KCL Misconduct Committee.

**You may find it easier to firstly generate a PDF of your source code using a text editor and then merge it to the end of your report. There are many free tools available that allow you to merge PDF files.**