KING'S
College
LONDON

University of London

6CCS3PRJ Final Year
[JK2] Automated Timeline Extraction

Final Project Report

Author: Oliver Philip Höhn

Supervisor: Dr Jeroen Keppens

Student ID: 1426248

April 15, 2017

Abstract

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 shows the events depicted in the documents through a timeline.

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 Philip Höhn

April 15, 2017

Acknowledgements

I would like to thank my supervisor, Dr. Jeroen Keppens, and King's College London. The supervision and support Dr. Keppens provided was extremely helpful throughout the progression of the project. Also, I would like to thank my family and friends for the continued support and encouragement throughout the project. I could have not done it without them.

Contents

Chapter 1

Introduction

The project aims to facilitate the understanding of a substantial number of documents (especially law related) through a graphical representation. When a law professional is tasked with a law case, they are expected to fully understand the overall structure and occurrence of the events described in the documents. However, when a large collection of documents is involved, this task can be both cumbersome for the employees and financially expensive for the employer. As the documents tend to be extensive and complex, requiring many working hours to understand the underlying story, it does not allow the professional to progress to the next stage of their work.

1.1 Project Scope

As the system requires analysing, processing and graphically representing the information in the documents, the main areas of the project are Natural Language Processing (NLP), Data Processing and Data Representation. There relevance to other works will be discussed in the Background Chapter.

1.2 Objectives

The Objectives are to produce a system that is effective, efficient, and simple to use. The system should require as input a selection of documents written in correct English (i.e. in natural language) and produce a timeline with the events described in the documents.

Since the user does not have to be computer scientists or have any technological knowledge, but rather are expected to be law professionals, the system should be approachable for them. Therefore, the visibility of what the system is doing, and what the users options are, is important.

The timeline should be self-explanatory. The user should understand which events happened during which time periods, and know from the information provided what occurred during an event.

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.

The system should allow users to change the input settings, such as the length of the summary, reference points or how many processors can the program use in parallel, to allow the user to see how the timeline changes with different input parameters.

The user may consider the need to save the timeline for later use, or further analyzation. Therefore, providing tools to save the data of the timeline as a PDF or as a JSON in the user's system would benefit them. Producing a JSON output would allow the system to be used by 3rd-parties, that may provide their own graphical user interface (UI) using the system to only process the documents; or further process the events identified by the system.

1.3 Report Structure

Following this chapter is a discussion of the background related to this project. Then a formal presentation of the requirements and specification of the project follow. Which is then followed by the design architecture and patterns used in the system. Followed by the implementation, testing and evaluation of the system. During the evaluation, it will be discussed to what extend 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 resulting system should aim 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 text that produced it.

For this system, the text of documents will be split into their sentences, which are analysed separately. An event is produced when a given sentence contains a date (or temporal expression). Events may contain a range of dates. A range is given by a start and end date. The range describes that the event occurred during that time period. For example, an event that occurred 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 on one specific day would only have one date associated with it.

The subjects of an event are given by the "person, place, thing, or idea that is doing or being something"[9]. These are key words that help convey the meaning of an event. They help the summary of the event, i.e. summary of the sentence that was identified as this event, by providing additional information used to understand what happened.

2.1 Natural Language Processing

The projects primary focus is on the field of Natural Language Processing (NLP). NLP is the field in Computer Science that focuses on producing systems that understand human readable and spoken language [20]. While there is ground breaking research in the field, many of the systems require the human input to be subject to constraints. As many of the systems use rule sets that are applied to the input to translate it to a useful input for the system. For example, removing ambiguities in text.

For this system, it is expected that the documents are written in correct English. This is because every language has their own grammar. The grammar can be used as a rule set for the language. Thereby using them to identify key words such as people, locations, and companies; and, using them to produce summaries in some case [7]. For example, it is to be expected that documents are written in correct English. To expand the system to different languages would require different rule sets to be applied depending on the language to process the text. Different languages would require different algorithms, models and rule sets used.

An issue that may occur in the input documents is that they are grammatically disorganised and incorrect. Many NLP software tools (including Stanford CoreNLP which is used in this project) perform extremely poorly with such input. One example is the performance of NLP tools with Tweets, short sentences used in the social media platform Twitter [1], that use informal abbreviations due to a character limit. In the paper Named Entity Recognition in Tweets: An Experimental Study [15], they looked at the performance of popular NLP tools on "Tweets". Many of the different NLP tools such as Apache OpenNLP and Stanford CoreNLP performed poorly. This is due to the NLP tools and their algorithms not being able to apply their rules on to the text to identify its different components. Hence, for this project it will be assumed that the input documents will be written in correct English. This is to be expected as the primary use of this tool is for formal documents, such as law documents.

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

2.1.1 Automatic Summarization

In Automatic Summarization, the aim is to produce a shorter version (a summary) of a given input text. The summary should still hold the same meaning of the original input. The summary can be built directly from the words in the input, or they can be built using a dictionary. Since in this project an event is built from a sentence that contains a date, the Headline Generation area of Automatic Summarization is focused on. In Headline Generation, a summary is built based on the data provided in the input text (its grammar), where a threshold value is given such that the aim is to provide a summary of that size. For Headline Generation, there are two main implementations: statistic based and decision (trimming) based [4].

In the statistic based model, Noisy-Channel models are the most prominent, as shown by

[1] https://twitter.com/

the multitude of publications [3, 4, 16]. 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 text and their summary), which are used in the calculation of the statistical values used to determine whether a produced summary correctly represents its original text. Examples of these algorithms can be found in the works of [4, 11].

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 [7]. This is done by tokenizing the sentence, i.e. breaking text into words, phrases, symbols and tagging them each by an identifier [12]. 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. These rules remove branches of the tree until no more rules can be applied or the summary text falls below a given threshold. The leaves of the trimmed tree are then used to produce the summary.

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 [11], the trimming based models can produce better summaries than the statistic models in some occasions. For example, statistic based models are domain dependent as their data set is for a specific domain. Thereby placing statistic models in another domain will cause them to perform poorly compared to the decision models that are domain-independent.

The main advantages of the trimming models are 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 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. Requiring a large data set causes algorithms to be processor-heavy because the annotated data needs to be read, and the statistical computations need to be carried out. This is not an issue when it is done for a large set of texts, but if it is done for every sentence identified with a temporal expression, then the computation-work is much larger than the input. Thereby, more work is being done for the computation than required.

Due to the time-constraints, it was decided to use the trimming approach, in specific the algorithm provided by Dorr, Zajic, and Schwartz [7] (see Algorithms 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 (given by the POS Treebank[2]), which is then trimmed. Note that removing a parent node, includes removing its children in the tree.

Algorithm 1: Dorr, B., Zajic, D. and Schwartz 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
get the leftmost-lower subtree with root S;
remove time expressions;
remove determiners (e.g. 'a', 'the');
/* Applying XP-Over-XP Rule                                              */
while the number of leaves in tree > threshold and there are subtrees to remove with this
  rule do
    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);
end
/* Applying XP-Before-NP Rule                                            */
while the number of leaves in tree > threshold and there are subtrees to remove with this
  rule do
    remove any XP (where XP can be NP, VP, PP) before the first NP found;
end
/* iterative shortening rule                                             */
get Tree T' from lastRule;
from T' create a sentence S by reading of the leaves in pre-order;
return S;
```

[2] https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

Algorithm 2: Last Rule

```
Input  : A Grammatical Tree T of the Sentence to summarize
Input  : threshold: Threshold value
Output: A Grammatical Tree T' of the Summary of the input after the last rule has
        been applied
if the number of leaves in tree > threshold then
    make a copy of the tree T, called T';
    while the number of leaves in tree > threshold and there are subtrees to remove with
      this rule do
        remove any trailing PP nodes (and their children);
    end
    if the number of leaves in tree > threshold then
        while the number of leaves in tree > threshold and there are subtrees to remove
          with this rule do
            remove any trailing SBARs (and their children);
        end
        while the number of leaves in tree > threshold and there are subtrees to remove
          with this rule do
            remove any trailing PPs (and their children);
        end
    end
    return T';
end
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[3]. Each word or set of words are given a part of speech tag identifying them. An example can be found below (see Figure 2.1).

On the grammatical tree shown in the figure, the algorithm proposed by Dorr, Zajic, and Schwartz [7] will be applied and the process will be graphically shown in the following figures (see Figures 2.2, 2.3, 2.3, 2.4 and 2.5). Note that the value of the threshold does not force the

[3] http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

summary to be below it, but it is a length of the summary to which the algorithm is working towards. 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 grammatical tree is produced 2.1.

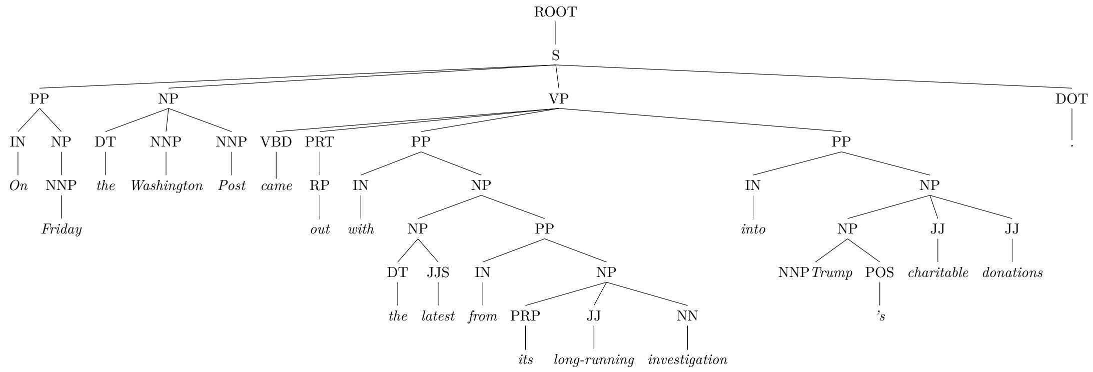


Figure 2.1: P.O.S/Grammatical Tree of: "On Friday the Washington Post came out with the latest from its long-running investigation into Trump's charitable donations."

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 or a direct identifier of a word. Using the Hedge-Trimmer algorithm [7], the tree is processed as follows. First, 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. The next step is to remove time expressions, of which there is only one, "Friday". However, when removing it, the "NP" parent must be removed, else there is a grammatically incorrect summary (given by the Hedge-Trimmer algorithm [7]). The result is graphically shown in the Figure 2.2.

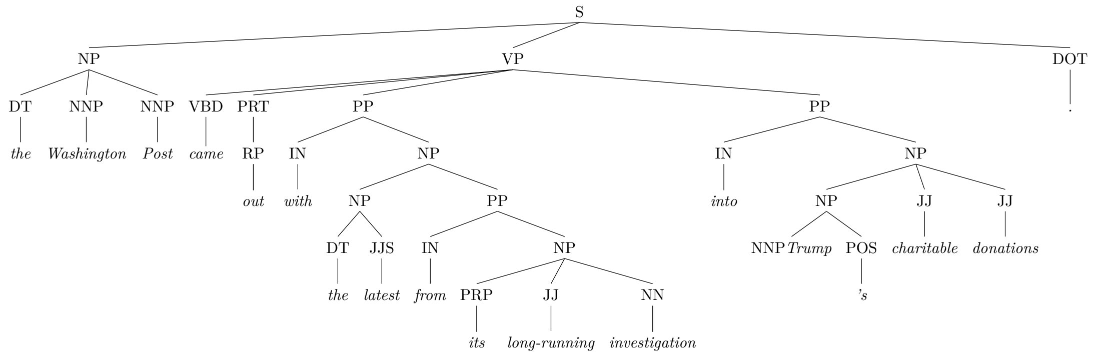


Figure 2.2: P.O.S/Grammatical Tree after removing time expressions

On the resulting tree, the algorithm dictates the removal of determiners. The "DT" parent tag identifies a determiner. However, not all the parent identifiers with this tag are removed, only ones that have a child labelled "the" or "a". The resulting tree after applying this rule is given by Figure 2.3.

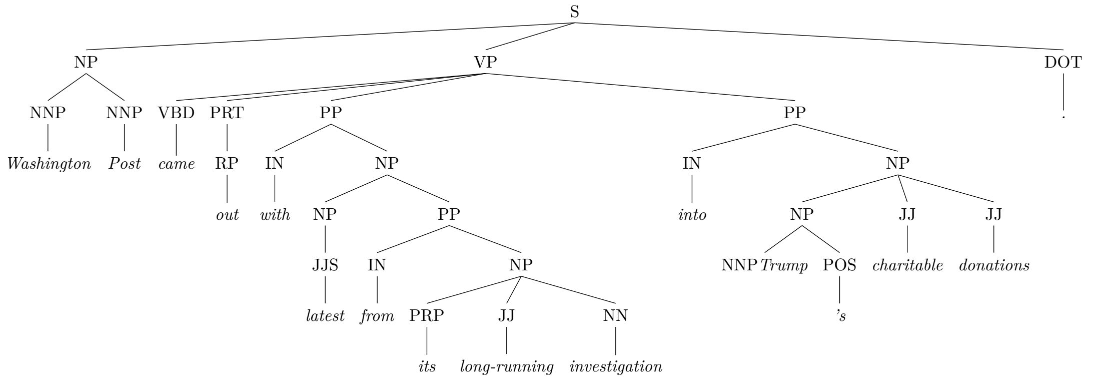


Figure 2.3: P.O.S/Grammatical Tree after removing determiners

The threshold value is used in the following rules. For this example, the threshold value is 10. From the two rules left, only one will be applied as the number of leaves in the resulting tree will fall below the threshold value. This avoids over-trimming the summary.

The threshold's value importance is of stopping the algorithm from over-trimming or under-trimming. This is done in the sense of there being an optimal summary where the core meaning is kept but if it is further reduced the meaning is lost. When the tree is over-trimmed, there is a possibility that the resulting summary does not have the core meaning of the original sentence, or has no meaning at all. 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 than the optimal summary. Thereby, having more words and thus retaining the original meaning, as it is 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. Under-trimming can ensure the meaning is kept in the summary, but not that the resulting summary is optimal in size (thereby incrementing the time required by the user to view the entire timeline, thus increasing the time the to understand what occurred, which is what the user aimed to avoid with the tool).

When applying the XP-Over-XP rule, if a parent identifier labelled XP (where XP can be NP, VP, or S) has a first child identifier of type XP, then all other children (except the first) of the parent are removed. This rule 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 and 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.

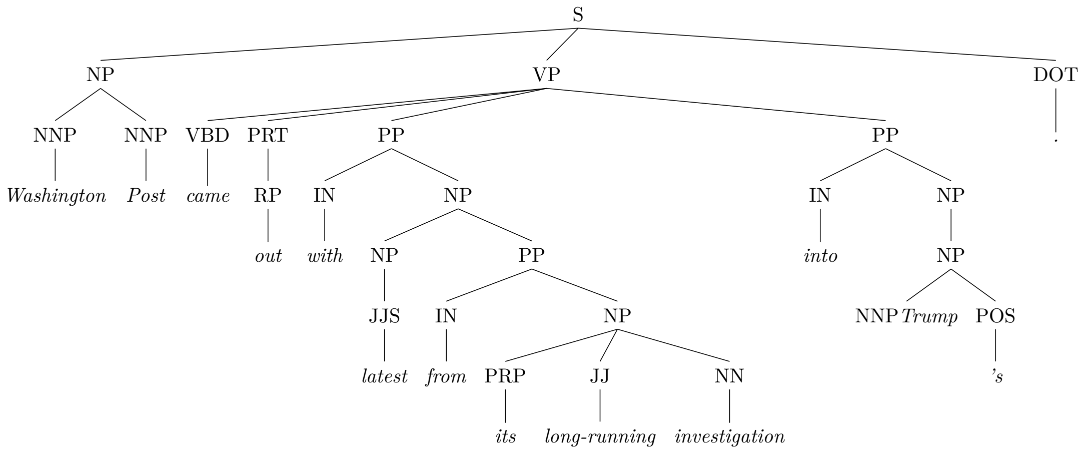


Figure 2.4: P.O.S/Grammatical Tree after XP-Over-XP first iteration

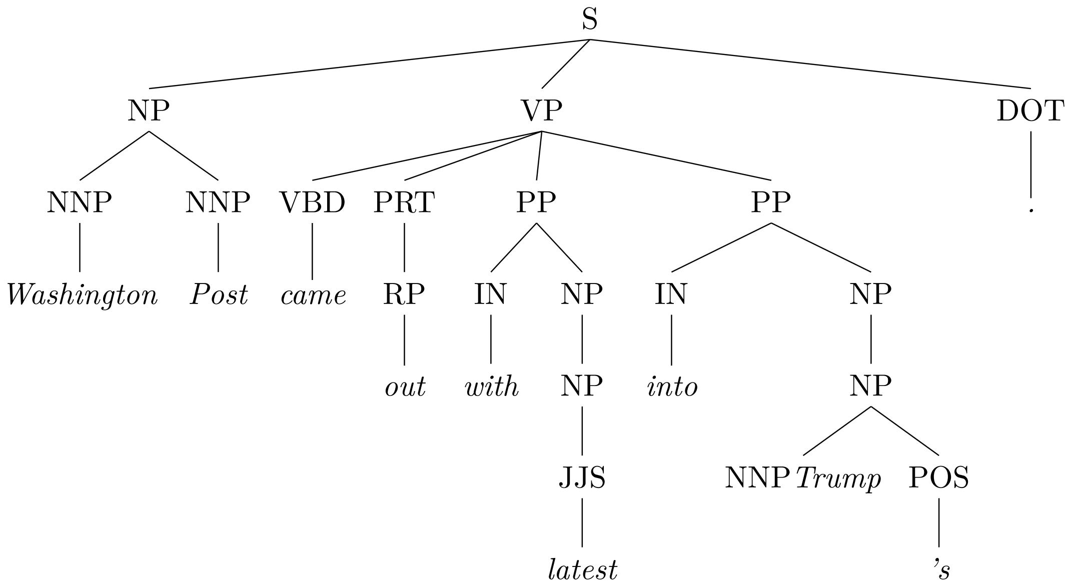


Figure 2.5: P.O.S/Grammatical Tree after XP-Over-XP second iteration

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. This rule consists of iteratively removing trailing PP and SBAR subtrees until the threshold value is reached, or the rule cannot be applied further.

The result of applying this algorithm (Hedge Trimmer [7]) on the input text: "On Friday the Washington Post came out with the latest from its long-running investigation into Trump's charitable donations.", produces a summary: "Washington Post came out with latest into Trump's". The original meaning of the sentence, which is that the Washington Post released a new article on Trump, is kept. However, 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 convey the meaning of the original sentence. This is a headline of the input, indicating what the input text was about, and thereby giving a general description of what occurred.

2.3 Normalizing Dates

An issue when processing documents, and identifying events, is that sentences will not always include the exact date of when an event occurred. These temporal expressions are ambiguous. For example, "Yesterday" or "Last week" have different meanings depending on the context in which they are in. To produce a timeline, exact dates are required to sort the events by their date. This is valuable to the user, as events that occur within the same time era will be grouped and separated from events of other time eras. To determine the exact date of an ambiguous temporal expression, a reference point can be used. For example, if the text was written on the 11th of March 2017, then "Yesterday" in this context refers to the 10th of March 2017, but if the document was written on the 1st 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.

The reference point (or base date) acts as a context of when an event occurred. Reference points, and similar techniques have been used in other timeline works (see [13]). 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 of an ambiguous temporal expression to be determined. 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. Therefore, the reference only helps the task of producing exact dates.

An issue that may arise is 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, in which an event occurred it can be determined reasonably [13] that it was somewhere between the start of 1980s, i.e. 01-01-1980, and the end of the 1980s, i.e. 31-12-1989. To produce an exact date, a range of start and end date can be used for this event. This allows for it to be compared to other events, to then sort, and inform the user the event occurred somewhere within that period.

Chapter 3

Requirements & Specification

3.1 Brief

The system should take as input a set of documents, process them autonomously (i.e. without the user's involvement), and produce a graphical representation of events in the documents. This requires the identification of sentences in the text that contain dates, providing an exact date (to compare to other events), identifying subjects and providing a summary of the sentence. Requirements are generated from this.

3.2 Requirements

3.2.1 Functional Requirements

The functional requirements of a system are behaviours a system should have[1]. 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.
6. Travel between timeline and relevant document.

[1] http://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/

7. Save timeline (as .pdf or .JSON).

As the software will require as input, documents, the 3 most used document file types[2] should be allowable file types in the system.

Since the aim is to identify events in the input text, and events are identified temporal expressions, then it is necessary to identify these.

The subjects of a sentence are required to aid the description of an event. Subjects, in this case, include names of people, locations, and quantities of money (i.e. key words).

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, providing the ability to edit the events would allow the user to correct these mistakes.

The final two requirements do not affect the processing of the system, but are advantageous to users. 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 (which is the original sentence of the event). Providing a save to PDF ability allows the timelines to be included in documents, as the PDF files can be merged. More interestingly, producing an intermediate JSON output makes the system compatible with 3rd-party applications that can provide other graphical representations and/or process further the data. Providing two graphical representations of a timeline (in the system and as a PDF) along with a JSON representation should allow the system to be integrated in documents, reports and other applications.

3.2.2 Non-Functional Requirements

Non-functional requirements of a project are descriptions of how the system must perform 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).

These three requirements will be evaluated to determine if they are met in the produced system.

Since the system should be used by any kind of user, with no required technical knowledge, it should be intuitive to the user to know how to use it, i.e. the system should be usable. The user of the system will be discussed later.

[2] http://www.computerhope.com/issues/ch001789.htm

It would be unreasonable that the resulting system is extremely slow. Such would be the case if on an input n it would require a much larger (exponential) time to complete.

Efficiency relates to the task of identifying events. Identifying 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.

To identify events, temporal expressions will need to be extracted and inferred. Extraction and inference of dates from temporal expressions will be discussed later. This issue is relevant as in some cases it is known that an event occurred after another, but the specific date cannot be determined. Thereby, producing a timeline of linked events, where an event appears after another not because its exact date suggests so, but because the context would allow for the events to be linked in such way. This requires changing models in established NLP tools, and thus will be discussed in the Future Works chapter.

3.3 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 must 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, which would not allow for the completion of the project. In addition, loading large models of data to summarize one sentence, would have a substantial impact in the running time of the system. From the users point of view, if the system is not significantly faster at producing timelines, then the user would prefer the more accurate manual timeline.

A limitation in all NLP projects is the technology. Modelling context in systems is extremely difficult and non-trivial [10]. For this project, it is a clear issue, that for some inputs, poor timelines will be produced as the context of the text was not fully modelled, and thus not understood by the system. The issue can be applied to ambiguous temporal expressions, where the exact date referenced in the sentence cannot be determined. Solutions to these problems include looking at sentences by relating them to each other (instead of independently), and thereby linking references. However, this still does not mimic the context understood by humans during read or spoken interactions.

3.4 Additional Aims

An aim of the project is to make it open-source by providing it on GitHub along with a license. The license would allow anyone to use the system. This is beneficial as the system can be integrated in other programs (directly, as a library, or indirectly, through JSON). All the libraries used allow for the system to be open-source.

Chapter 4

Design

For the design, a clear set of objectives, use cases and architecture have been developed to aid the implementation.

4.1 Objectives

The Objectives of the system are for it to be visible, efficient, and effective.

Visible - It should be visible to the user what functions are available. This includes being able to distinguish actions from informative text. Thereby, providing an application that can be picked up and used with minimal to no training and benefiting the growth of users of the system. This can be achieved by providing a simple and intuitive User Interface (UI), where buttons are highlighted, and actions are not cluttered, instead only necessary actions are provided.

Efficient - The time spent to perform tasks should be reasonable to the context of its input. This can be further expressed in the 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 (visually shown as $O(n)$). This allows the efficiency to be compared between algorithms, as the focus is on how well they scale with larger inputs. The objective is to not have an algorithm that processes the files with an exponential time complexity. With an exponential time complexity, a small n would lead to a large running time, therefore a larger n would result in an infeasible running time. Threads can be used to aid this task. 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 all documents are being processed in parallel, completely independently of each other, then the time of the document that takes the longest to process would result in the time it takes to process the entire set of documents.

Effective - The system should meet its purpose. Which is to require as input, documents, and process them to produce a timeline. The system should allow the user to perform these tasks through menu options, buttons, and, in addition, produce appropriate responses. When an error occurs, the system should not attempt to process the documents indefinitely, and instead produce a timeline with the available events. Effectiveness also involves how correct are the timelines produced by the system. Whether they provide the user the correct information, and if the events are being identified in the documents.

4.2 Use Cases

A use case is a task an actor in the system performs. An actor is any type of user of the system. In this case, the user can be a law professional that uses the system to have a general understanding of a set of documents. Therefore, it can be assumed that the user does not necessarily have experience with NLP. It should be transparent to the user how the documents are being parsed, and only if they are interested would they require looking 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 view and interact with the produced timeline. In some cases, the user may produce their own graphical representation of a timeline and just use the produced JSON of the system. This would be the case if they have developed their own system to interact with this system.

The requirements give the use cases of the system, and they are presented below.

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.

The main sequence are the steps of the interaction in the use case to reach the goal. An error can occur during the interaction, it should be the systems responsibility to deal with the error appropriately, and not end the execution of the program. The primary actor, is the agent or entity that initiates the use case most of the time [2].

Note that loading documents 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 they originate from. Reference dates are included in the check of duplicate events as it may be the case that different events are produced for the same document when different reference points are used.

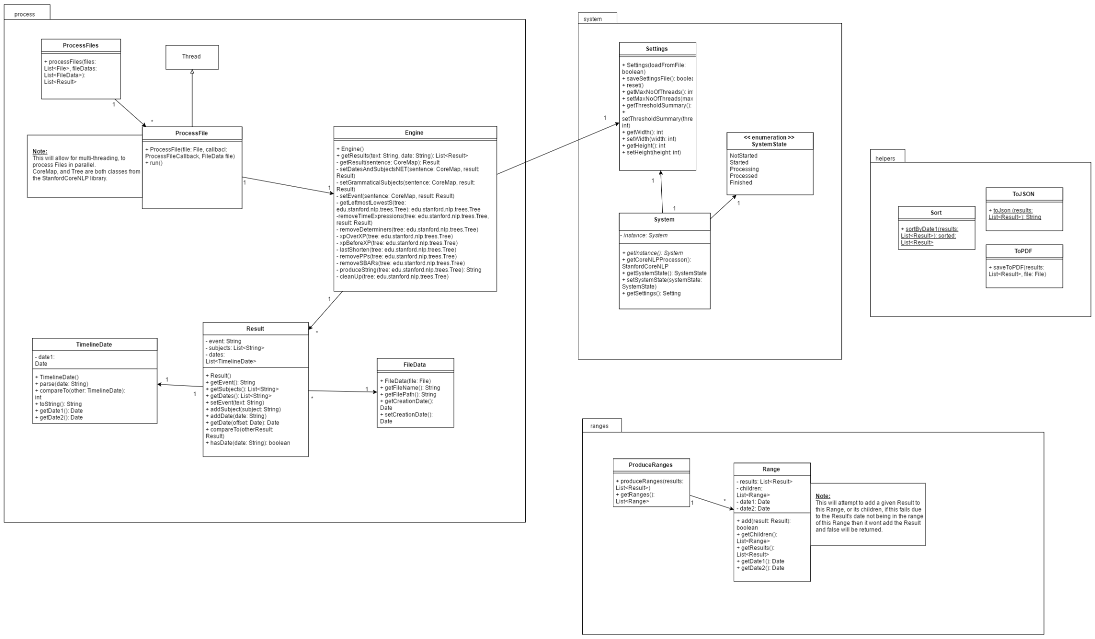
When an event is being edited, the user should have the option to delete it, as it may be that the system produced an erroneous event or the event is not relevant to the user. Events may not be relevant to the user if the same event occurs in two separate documents. This can occur when the events described in the documents overlap. The duplicate check would not identify them as duplicates as they originate from different files. To ensure the date entered by the user is correct, it is validated. The validation checks are carried out before the changes are saved. An example of invalid data is when the date of an event is modified but instead of a new date being set, other text data is entered. In the case of the event occurring during a range of dates, i.e. it has a start date and end date, a validation check should be that the second date does not occur before the first. The checks should be carried out before the data is saved. In the case the validation fails, the changes should not be changed, instead the user should be prompted to correct their input.

During the saving use case, the desired location can be unavailable. This can be either because the Operating System does not allow the application to write to that directory, or because a file that is in use is being overwritten (i.e. a lock has been placed on it). Therefore, the user should be prompted to save in another location.

4.3 Architecture

The architecture of a system, is the structure of the components in the system[1]. The focus is on the back-end, or logic, of the system as opposed to the graphical, or front-end. 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 4.1). Each package will be looked at individually.

Figure 4.1: Full UML Architecture of Automated Timeline Extraction system.


process
ProcessFiles
+ processFiles(files: List<File>, fileDatas: List<FileData>): List<Result>
Thread
1
*
Note:
This will allow for multi-threading, to process Files in parallel.
CoreMap, and Tree are both classes from the StanfordCoreNLP library.
ProcessFile
+ ProcessFile(file: File, callbacl: ProcessFileCallback, FileData file)
+ run()
1
1
Engine
+ Engine()
+ getResults(text: String, date: String): List<Result>
- getResult(sentence: CoreMap): Result
- setDatesAndSubjectsNET(sentence: CoreMap, result: Result)
- setGrammaticalSubjects(sentence: CoreMap, result: Result)
- setEvent(sentence: CoreMap, result: Result)
- getLeftmostLowestS(tree: edu.stanford.nlp.trees.Tree): edu.stanford.nlp.trees.Tree
-removeTimeExpressions(tree: edu.stanford.nlp.trees.Tree, result: Result)
- removeDeterminers(tree: edu.stanford.nlp.trees.Tree)
- xpOverXP(tree: edu.stanford.nlp.trees.Tree)
- xpBeforeXP(tree: edu.stanford.nlp.trees.Tree)
- lastShorten(tree: edu.stanford.nlp.trees.Tree)
- removePPs(tree: edu.stanford.nlp.trees.Tree)
- removeSBARs(tree: edu.stanford.nlp.trees.Tree)
- produceString(tree: edu.stanford.nlp.trees.Tree): String
- cleanUp(tree: edu.stanford.nlp.trees.Tree)
1
*
TimelineDate
- date1: Date
+ TimelineDate()
+ parse(date: String)
+ compareTo(other: TimelineDate): int
+ toString(): String
+ getDate1(): Date
+ getDate2(): Date
1
1
Result
- event: String
- subjects: List<String>
- dates: List<TimelineDate>
+ Result()
+ getEvent(): String
+ getSubjects(): List<String>
+ getDates(): List<String>
+ setEvent(text: String)
+ addSubject(subject: String)
+ addDate(date: String)
+ getDate(offset: Date): Date
+ compareTo(otherResult: Result)
+ hasDate(date: String): boolean
*
1
FileData
+ FileData(file: File)
+ getFileName(): String
+ getFilePath(): String
+ getCreationDate(): Date
+ setCreationDate(): Date
system
1
Settings
+ Settings(loadFromFile: boolean)
+ saveSettingsFile(): boolea
+ reset()
+ getMaxNoOfThreads(): in
+ setMaxNoOfThreads(max
+ getThresholdSummary():
+ setThresholdSummary(thre int)
+ getWidth(): int
+ setWidth(width: int)
+ getHeight(): int
+ setHeight(height: int)
<< enumeration >>
SystemState
NotStarted
Started
Processing
Processed
Finished
1
1
1
System
- instance: System
+ getInstance(): System
+ getCoreNLPProcessor(): StanfordCoreNLP
+ getSystemState(): SystemState
+ setSystemState(systemState: SystemState)
+ getSettings(): Setting
helpers
Sort
+ sortByDate1(results: List<Result>): sorted: List<Result>
ToJSON
+ toJson (results: List<Result>): String
ToPDF
+ saveToPDF(results: List<Result>, file: File)
ranges
ProduceRanges
+ produceRanges(results: List<Result>)
+ getRanges(): List<Range>
1
*
Range
- results: List<Result>
- children: List<Range>
- date1: Date
- date2: Date
+ add(result: Result): boolean
+ getChildren(): List<Range>
+ getResults(): List<Result>
+ getDate1(): Date
+ getDate2(): Date
Note:
This will attempt to add a given Result to this Range, or its children, if this fails due to the Result's date not being in the range of this Range then it wont add the Result and false will be returned.

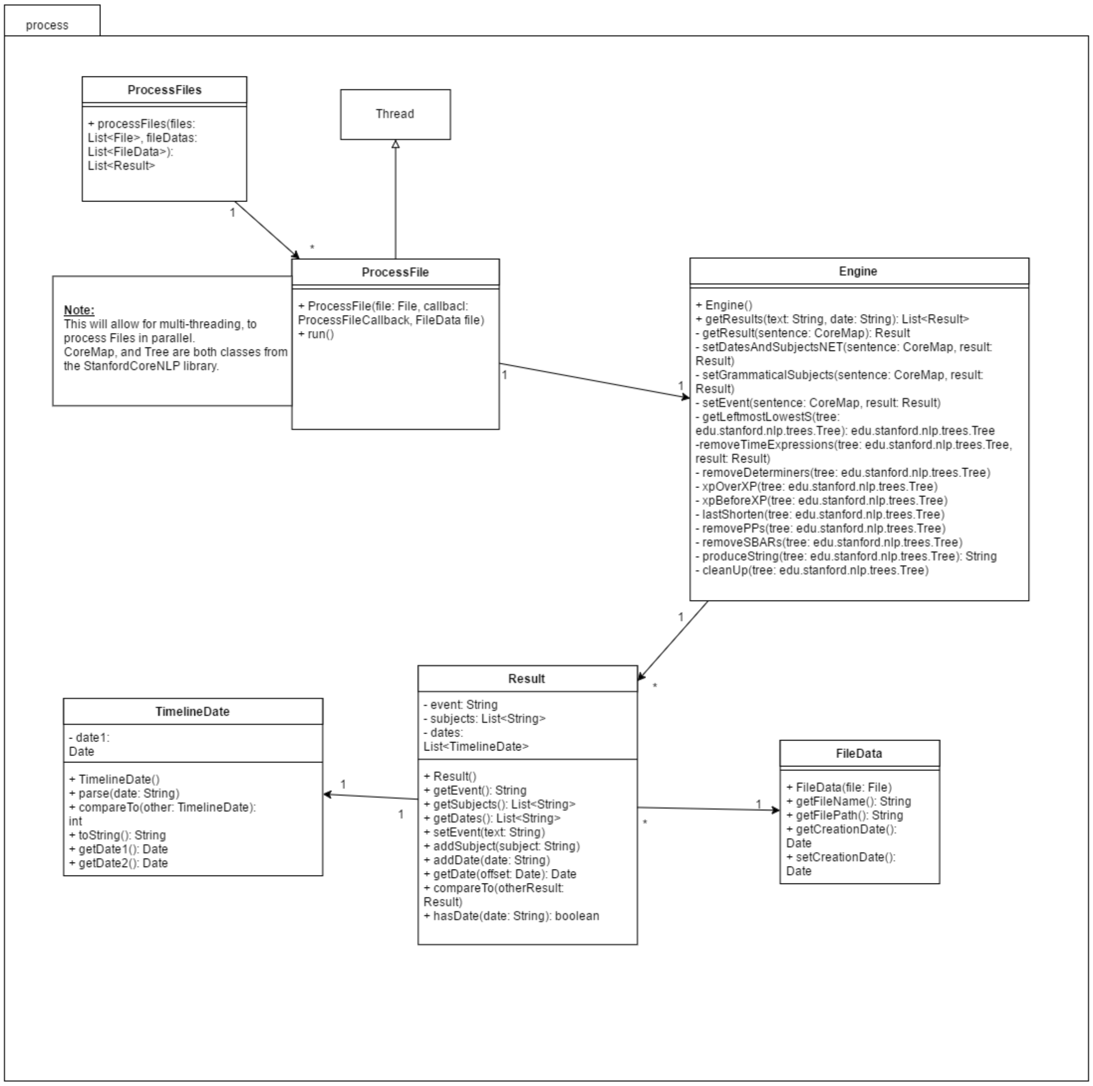
4.3.1 Process Package

The core of the system is the process package (see Figure 4.2). The architecture used here is Business Delegate, where all the interaction to the package is through one component, ProcessFiles, which delegates the work to the other components in the package. A list of files is passed in, along with their data (such as the file name, its path, or the creation/reference date used). Each file is processed in parallel. The maximum number of parallel processing allowed is given by the settings of the system (in the settings package). This is the maximum

[1] `https://msdn.microsoft.com/en-gb/library/ee658098.aspx`

number of threads that can be ran at any given point. However, in the implementation one more thread should be added to the count, as the graphical user interface always runs on a separate thread. The belief is that with a maximum setting of n threads, that 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 can be processed 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 the implementation, semaphores can be used to aid this task. A semaphore is a data structure that limit how many threads run by requiring threads to acquire a lock before performing their parallel processing. The thread afterwards releases the lock. If no locks are available, the thread waits until a lock is available.

Figure 4.2: UML of the Processing Package


process
ProcessFiles
+ processFiles(files: List<File>, fileDatas: List<FileData>): List<Result>
Thread
1
*
Note:
This will allow for multi-threading, to process Files in parallel.
CoreMap, and Tree are both classes from the StanfordCoreNLP library.
ProcessFile
+ ProcessFile(file: File, callbacl: ProcessFileCallback, FileData file)
+ run()
1
1
Engine
+ Engine()
+ getResults(text: String, date: String): List<Result>
- getResult(sentence: CoreMap): Result
- setDatesAndSubjectsNET(sentence: CoreMap, result: Result)
- setGrammaticalSubjects(sentence: CoreMap, result: Result)
- setEvent(sentence: CoreMap, result: Result)
- getLeftmostLowestS(tree: edu.stanford.nlp.trees.Tree): edu.stanford.nlp.trees.Tree
-removeTimeExpressions(tree: edu.stanford.nlp.trees.Tree, result: Result)
- removeDeterminers(tree: edu.stanford.nlp.trees.Tree)
- xpOverXP(tree: edu.stanford.nlp.trees.Tree)
- xpBeforeXP(tree: edu.stanford.nlp.trees.Tree)
- lastShorten(tree: edu.stanford.nlp.trees.Tree)
- removePPs(tree: edu.stanford.nlp.trees.Tree)
- removeSBARs(tree: edu.stanford.nlp.trees.Tree)
- produceString(tree: edu.stanford.nlp.trees.Tree): String
- cleanUp(tree: edu.stanford.nlp.trees.Tree)
1
*
TimelineDate
- date1: Date
+ TimelineDate()
+ parse(date: String)
+ compareTo(other: TimelineDate): int
+ toString(): String
+ getDate1(): Date
+ getDate2(): Date
1
1
Result
- event: String
- subjects: List<String>
- dates: List<TimelineDate>
+ Result()
+ getEvent(): String
+ getSubjects(): List<String>
+ getDates(): List<String>
+ setEvent(text: String)
+ addSubject(subject: String)
+ addDate(date: String)
+ getDate(offset: Date): Date
+ compareTo(otherResult: Result)
+ hasDate(date: String): boolean
*
1
FileData
+ FileData(file: File)
+ getFileName(): String
+ getFilePath(): String
+ getCreationDate(): Date
+ setCreationDate(): Date

```
Algorithm 3: Algorithm for processing a list of Files
Input   : A list of Files to Process
Output: A list of Results
foreach File in the input list do
    wait until can run;
    /* if the maximum number of threads running in parallel has not been
       reached then stop waiting, else wait                                   */
    process the file;
    add the produced Result to the list of Results to return;
end
return list of Results;
```

For this system, an event is described as a Result that contains date information (represented by TimelineDate), a set of subjects (or key words), and the summary of the sentence that produced the event.

The TimelineDate component processes temporal expressions identified by the Engine, these are parsed and an exact date is produced which is then used to compare and sort the Results (events).

In this system, the Stanford CoreNLP suite[2] is used. It is a well-known and tested tool for NLP. Other tools exist, such as Apache OpenNLP, however Stanford's tool has a larger set of documentation and support, as well as being thread-safe and efficient. 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, which is then parsed through the Engine. The Engine is responsible for producing the Results for a file. It identifies sentences with dates, and for those extracts subjects such as people, locations, money, etc. It uses the Hedge-Trimmer algorithm, discussed in the Background Chapter, to produce a summary.

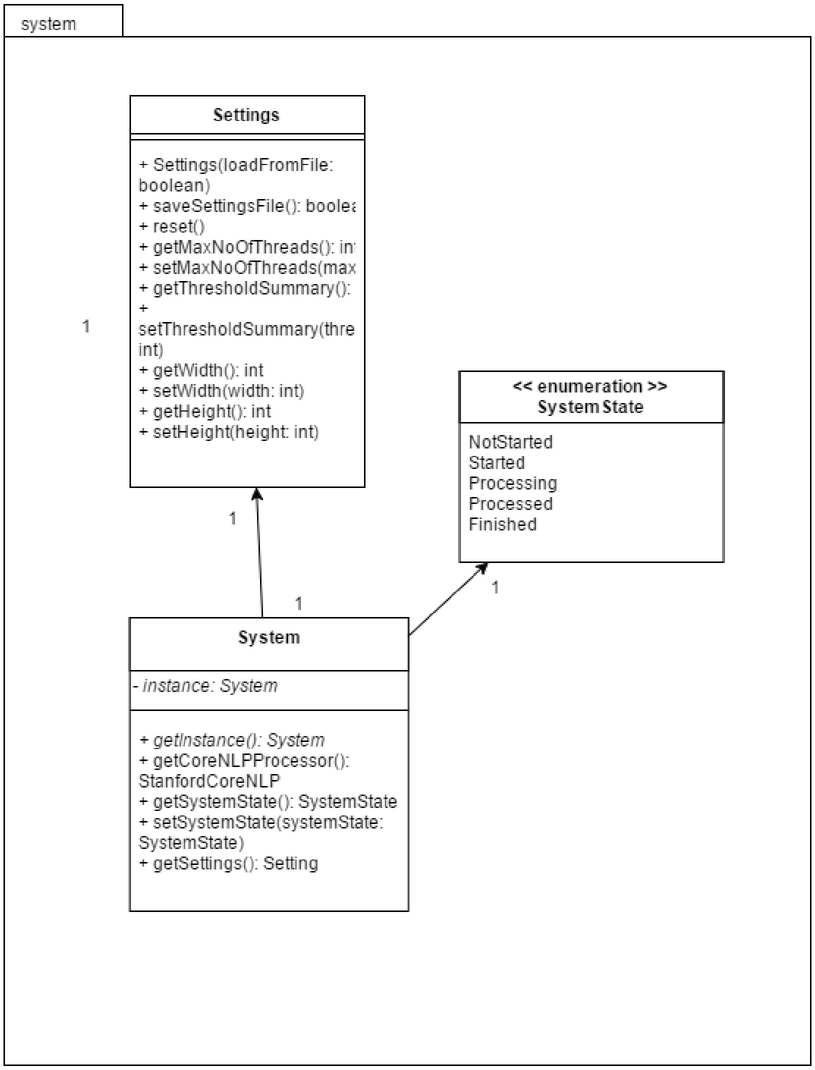
4.3.2 System Package

The system package holds the system and settings components (see Figure 4.3). 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 summary algorithm, and graphical settings. The SystemState attribute is used to identify at which stage the system is in when it is processing files. This

[2] http://stanfordnlp.github.io/CoreNLP/

allows for the logic of the system to be decoupled from the graphical representation, as it can use the system state to determine which actions are available, when a loading bar needs to be shown, and when the timeline is ready to be displayed. The component is shared throughout the system, as it contains data required in different components.

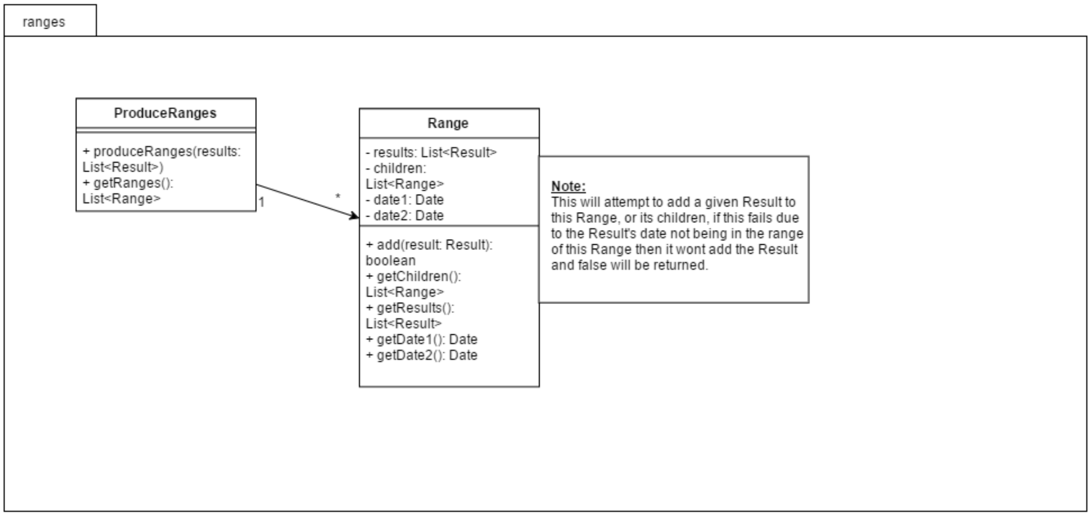
Figure 4.3: UML of the System Package



4.3.3 Ranges Package

The ranges package was the last package developed (see Figure 4.4). It focuses on placing Results into ranges (see the Algorithm 4), hence the name. A start and end date define a Range. 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 is attempted to place the Result in one of the children of this Range. If this is not possible, then another Range root is checked. Since Ranges are similar to Trees (as they recursively include other Ranges), there may be a forest of Ranges in the system. The roots are Ranges with the largest range of dates that encapsulates its child Ranges (and their Results). Thereby allowing to group related events together, and encapsulating them by their dates.

Figure 4.4: UML of the Ranges Package


ranges
ProduceRanges
+ produceRanges(results: List<Result>)
+ getRanges(): List<Range>
1
*
Range
- results: List<Result>
- children: List<Range>
- date1: Date
- date2: Date
+ add(result: Result): boolean
+ getChildren(): List<Range>
+ getResults(): List<Result>
+ getDate1(): Date
+ getDate2(): Date
Note:
This will attempt to add a given Result to this Range, or its children, if this fails due to the Result's date not being in the range of this Range then it wont add the Result and false will be returned.

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
sort the list of Results by the number of days in between the start and end date in
  descending order;
foreach Result in the sorted list do
    attempt to add it to one of the existing Range roots;
    if failed to add to existing Range then
        make a new Range using the data of the Result;
        add the new Range to the list of Range roots;
    end
end
return list of Range roots;
```

The algorithm proceeds as follows. The Results are sorted by their range, i.e. the number of days between their start and end date. A Result that only has one date, has a range of 0. The Results with the largest ranges are added first as it is more likely that they encapsulate other Results (does not apply the other way around). This leads to the production of a tree, shown below. The root is a Range with a start and end date that encapsulates all the dates of its child Ranges. It may be necessary to expand the dates of a Range if it is the case that the dates of a Result partially overlap a Range. An expanded Range has no results, but has two children: the newly made Range for the Result that was being added, and the Range that the Result was previously partially overlapping.

To demonstrate the algorithm an example will be presented with the two Results presented in the table 4.5 and the pre-existing tree in Figure 4.6. First, the Results would be sorted by their range, such that the second result is the first to be added. This produces the tree in Figure 4.7. Then the next Result in the table is added, producing the tree in Figure 4.8. For the resulting tree, it was determined that the Result being added overlapped an existing Range, thereby a new Range was formed that would encapsulate the previous Range and the result being added. This is done by extending the start and end dates appropriately. To place the Result in the tree after the Range was expanded, a new Range was created to hold it, and the previous Range that was partially being overlapped is now a child of the expanded Range.

As can be seen from the tree produced, the largest start-end date pair encapsulates the smaller start-end date pair, this can be done recursively. This allows for a graphical representation where the events are encapsulated by their dates, providing an alternative view to the traditional top-down timeline.

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

Figure 4.5: Example Results Start and End Date

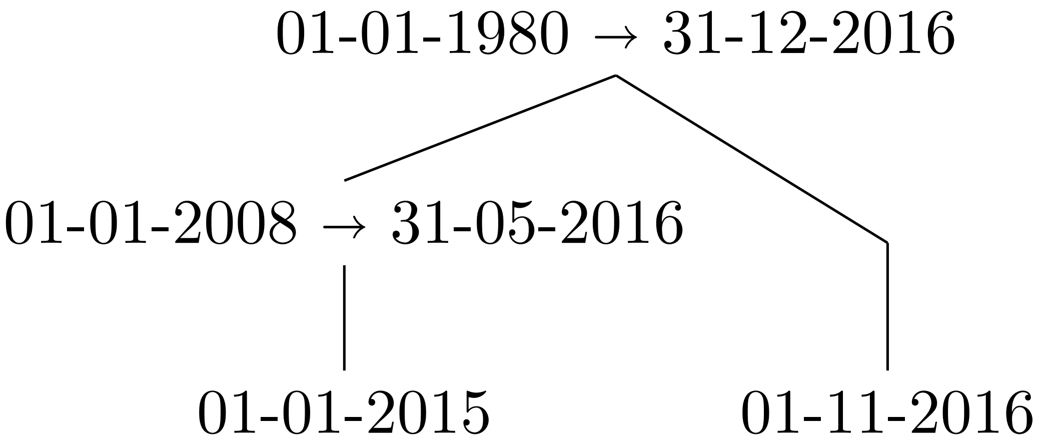


Figure 4.6: Pre-existing Range Tree

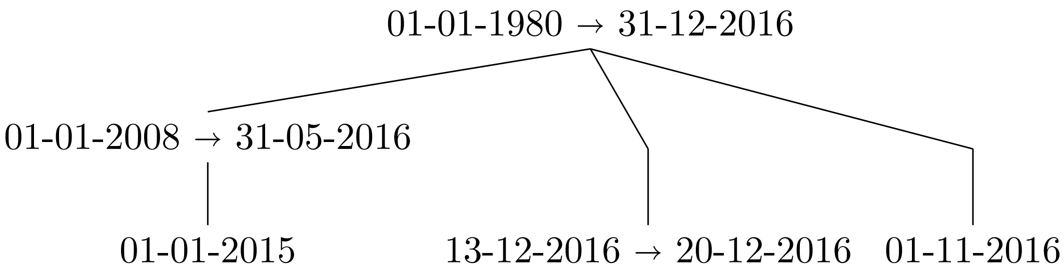


Figure 4.7: Resulting Range Tree after adding the Result: 13-12-2016 → 20-12-2016

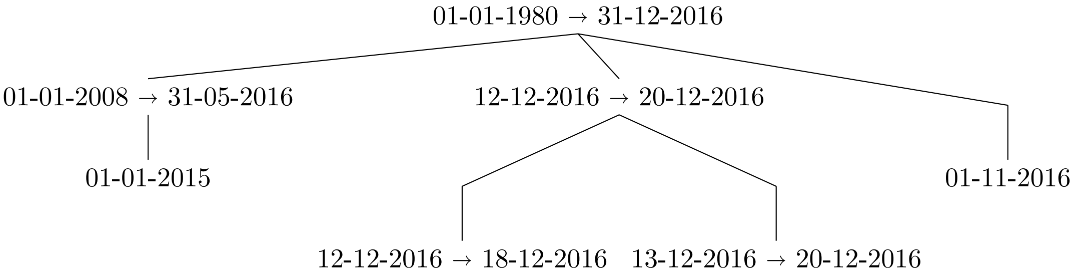
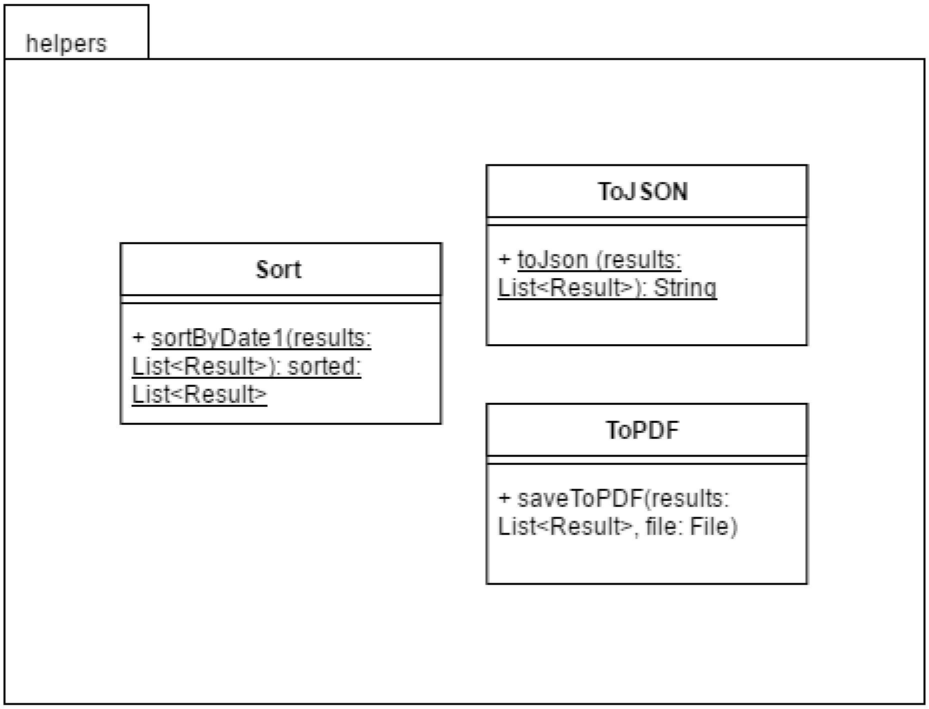


Figure 4.8: Resulting Range Tree after adding the Result: 12-12-2016 → 18-12-2016

4.3.4 Helpers Package

The helper package holds utility components required throughout the system (see Figure 4.9). For example, it provides sorting functions (sorting by the start date or by the range of days in between the start and end date) and saving the timeline in PDF or JSON format. It is a refactored package of reused logic in the system. Refactoring is done to remove repeated components and operations. Thereby having them in one place, 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 duplicated occurrence.

Figure 4.9: UML of the Helpers Package


helpers
ToJSON
+ toJson (results: List<Result>): String
Sort
+ sortByDate1(results: List<Result>): sorted: List<Result>
ToPDF
+ saveToPDF(results: List<Result>, file: File)

4.4 Design Patterns

Design Patters are general solutions to common problems in software development[3]. The solutions usually include an 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[4]. In most cases the number of components is restricted to one. The advantage of this pattern is to provide, a universal access for global data throughout the system. For example, in the System component the NLP processing tool is held, along with the system state. These are both used throughout the system, hence it would reasonable to use a Singleton pattern to ensure their availability, and not reload them into memory each time.

Observer - This design pattern allows a subject to notify its list of observers (or dependents). The subject can be a model, and 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 system states available.

[3] https://sourcemaking.com/design_patterns

[4] http://www.journaldev.com/1377/java-singleton-design-pattern-best-practices-examples

As the back-end begins processing documents, its state changes. The front-end should then change appropriately in such case, by retrieving the relevant data and showing it for that state. If the system were to be developed further, with other graphical interfaces used, then these could be added effortlessly, as 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 like the Observer design pattern, in that it separates the view from the data (which is a model) through a controller. The controller manipulates the data, and applies the appropriate changes to 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 (JavaFX[5]).

Other design patterns are available; however, these are the most relevant ones for this project. While the solution to ensure safe multi-threading is not a design pattern, it is worth mentioning as it allows for safe independent processing of documents using semaphores.

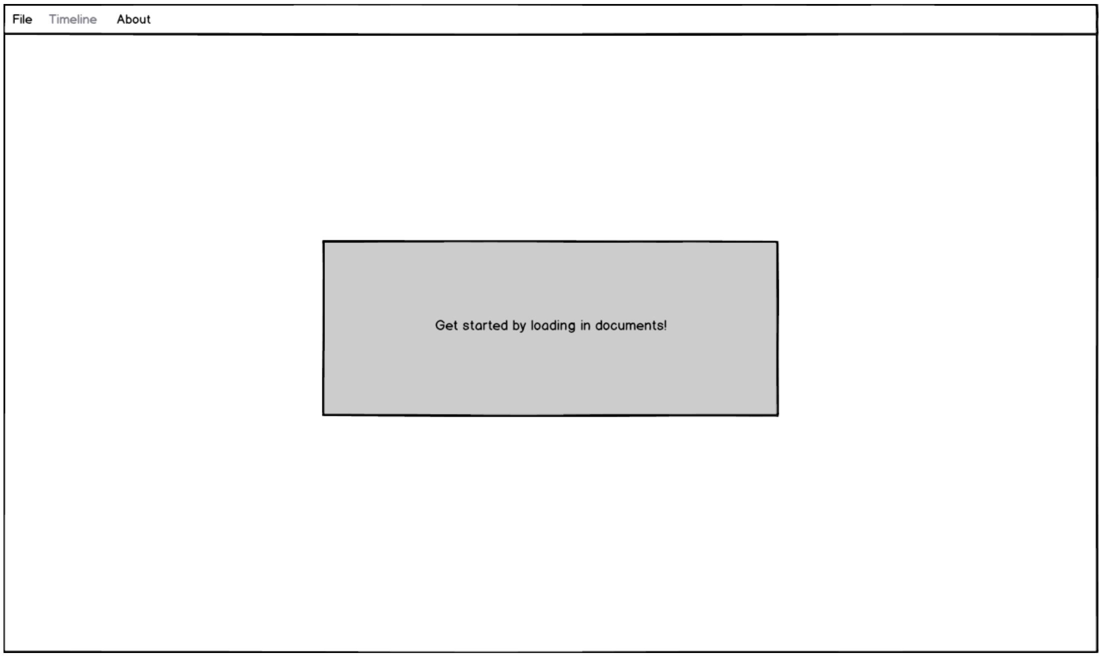
4.5 UI

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. This is considered during the development of the User-Interface. For example, as the main data representation of the system is the timeline of events, it should occupy much of the screen. Options that are not relevant to the current state of the system should not be available, while options that are most likely to be used should be highlighted. The aim is to build a UI that is visible and easy to use for the user.

Actions that are made unavailable ensure that the user is not overloaded with buttons and menu options that do not make sense within the current state of processing the system is in. For example, when the system is initially launched, no timeline can be presented as no documents are available to be processed. Instead the user is invited to load documents, as can be seen by the Wireframe in Figure 4.10. Wireframes are colourless screenshots of what the UI of a system should aim to look like.

[5] http://docs.oracle.com/javase/8/javase-clienttechnologies.htm

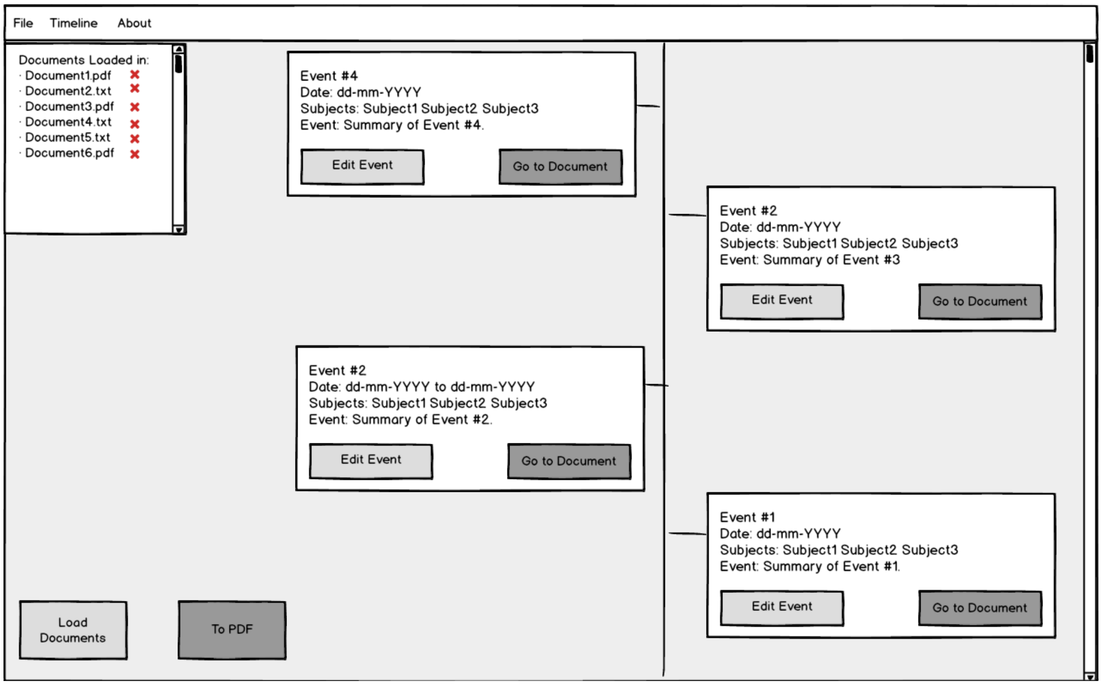
Figure 4.10: Wireframe of System at Start-up



Get started by loading in documents!

From the start up screen, the user can load documents and set the reference points used for the documents through a dialog displayed after the files have been selected. As this is being done, the system can load any resources it requires (such as models) before the files are to be processed. After the documents have been processed, the timeline is displayed (see Figure4.11).

Figure 4.11: Timeline of System



Documents Loaded in:
- Document1.pdf ✕
- Document2.txt ✕
- Document3.pdf ✕
- Document4.txt ✕
- Document5.txt ✕
- Document6.pdf ✕

Event #4
Date: dd-mm-YYYY
Subjects: Subject1 Subject2 Subject3
Event: Summary of Event #4.

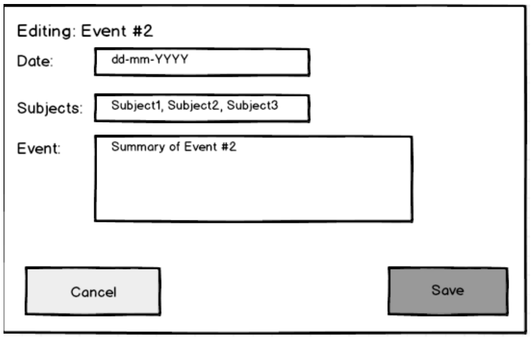
Event #2
Date: dd-mm-YYYY
Subjects: Subject1 Subject2 Subject3
Event: Summary of Event #3

Event #2
Date: dd-mm-YYYY to dd-mm-YYYY
Subjects: Subject1 Subject2 Subject3
Event: Summary of Event #2.

Event #1
Date: dd-mm-YYYY
Subjects: Subject1 Subject2 Subject3
Event: Summary of Event #1.

For each event produced by the system, a timeline row will be produced. With each event, options to edit (which will open a dialog, as seen in Figure 4.12), and view the sentence that produced the event in its own context (see Figure 4.13), are provided. The user is also provided with additional information of which documents have been loaded, along with the ability to load more or remove some of the documents, and save the timeline. Note that the timeline menu item is unavailable when there is no timeline displayed (see Figure 4.10), but when one is displayed it is available (see Figure 4.11).

Figure 4.12: Edit Event Dialog of System



Editing: Event #2

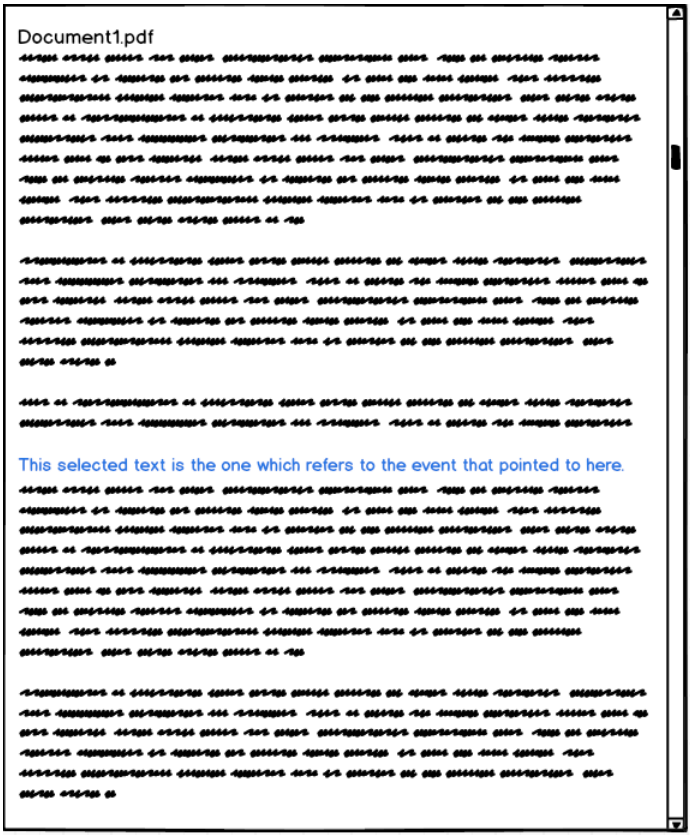
Date: dd-mm-YYYY

Subjects: Subject1, Subject2, Subject3

Event: Summary of Event #2

In the Document Viewer (see Figure 4.13), note 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 it, giving them a better understanding of what occurred. This also allows the user to swap between the system and the documents. In addition, it allows the user to view how each event has been produced and judge for themselves how effective the system is. The user can also jump through the document using the timeline by selecting the view function from an event. Therefore, the user is no longer required to read the entire document or to scroll through it if they are only interested in certain areas of it. This is extremely beneficial for large documents which are cumbersome to read, as the user can focus on specific areas 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. Note the tool should not replace the reading of highly sensitive and critical law documents (or any other kind of documents), as the user may require the full context of the document, but it can serve as a time-effective tool to skip through the document event-by-event, focusing on certain aspects of the document being revisited.

Figure 4.13: Document Viewer of System



Document1.pdf

This selected text is the one which refers to the event that pointed to here.

Chapter 5

Implementation

5.1 Approach

The development approach focused on the business-logic, or back-end, of the system. From the two main development methodologies, Waterfall and Agile, the latter was used. In Waterfall, the development process is sequential. Development is a sequence of phases that are completed one after the other. When a stage is completed it is not returned to, as like with a Waterfall, it is not possible to go up to the previous stage until the other stages have been completed. Agile development focuses on adaptive planning, evolutionary development, and continuous improvement. The advantage of using an Agile approach over a Waterfall approach is that unpredicted features and issues are dealt with when they arise. This does make Agile difficult to manage, as there is no clear indication of when the project will be completed.

Scrum is an Agile approach that uses Sprint cycles, which are short development periods. In the development cycles, a set of features must be implemented and tested. Testing allows to ensure that when new features are developed, the older features continue to work as expected. At the end of each cycle, meetings are held discussing what are the next steps, and what issues arose during the previous cycle. For this project, the meetings were with the supervisor.

The Agile approach proved to be useful when a new timeline view had to be implemented. A clear example of the advantage of this system was when a new timeline view was suggested. In this new view, events are grouped by their start and end dates, such that events that occur within the period of other events are encapsulated by the longer event. 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.

Note that the Agile methodology is used in software development teams. However, it can be applied to individual development, since the structure of the project allows for sprint cycles, review meetings with the supervisor, and the possibility of new requirements.

5.2 Tools & Software Libraries

The development environment of the project includes a 64-bit Windows 10 machine, with an 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 to be separated from the current working version of the code through the usage of branches. New features are added to the working version of the code, only if the required tests pass. Note this follows a Git flow development. In Git flow, a develop branch is made with the newest working features of the system, and new features are developed on separate branches. A master branch will only contain the latest fully implemented working version of the product. Using branches allows for bugs and development mistakes to be pin-pointed to the development of a specific feature, and allows the rollback to a previous, bug-free, state.

Gradle allows for libraries to be regarded as dependencies of the project. When the system is running on a separate machine, Gradle will retrieve 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 user's system when they run the command:

gradlew run.

Where gradlew is a wrapper for Gradle, such that the user does not even have to have Gradle installed in their system to run the application. Thereby, providing advantages in portability and general use.

As mentioned in the Background chapter, multiple libraries exist to aid the task of NLP. These are especially needed for the Named Entity Recognition(NER) and Text Summary tasks. An NER annotator will tag certain words, or collection of words into predefined categories such as people, companies, locations, and money. These tools also aid in tagging words using the POS Treebank, which is a requirement to use the Hedge-Trimmer algorithm [7] to produce summaries. Using an NLP library speeds up the production of the system, as manually building an annotator requires multiple developers and years of work. As can be seen from the release history of the Stanford CoreNLP tool, development began in 2010 and the latest release was in 2016[1]. The two main NLP tools available are Apache's OpenNLP[2] and Stanford's CoreNLP[3]. For this project, the Stanford's tool was used in the implementation, as it provides an extensive documentation, and examples, along with specific sections describing their annotators in detail.

The Stanford tool is the main library used throughout the project, as the project is reliant on its NER [8] and POS [18] annotators. It comes with models, that are loaded during the initialisation of the system. These models are used in the statistical calculations in the annotators to determine whether certain words fall into predefined categories, or which POS identifier tags should be given to them.

Due to the two main NLP libraries available being 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 making 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 to be tested. Testing is used in Git flow, to ensure that new features developed do not affect the correctness of older features, and are only merged together if the test succeed. Thereby, ensuring the system functions as expected.

Libraries for text extraction of .pdf and .docx file types are required, as the encoding of these files is not in plain text. The Apache POI[4] and the Apache PDFBox[5] are used. In addition to text extraction, the PDFBox library along with the Apache Commons library allows the creation of PDFs. This allows to save timelines as PDF documents. The Google GSON[6] library is used for the creation of JSONs, to save the timeline in a JSON file. The RichTextFX[7] library along with JavaFX library are used to build the Graphical User Interface (GUI) of the system. It was ensured that the libraries used have licenses that allow their use in this project, and its later release to open-source.

[1] http://stanfordnlp.github.io/CoreNLP/history.html

[2] https://opennlp.apache.org/

[3] http://stanfordnlp.github.io/CoreNLP/index.html

[4] https://poi.apache.org/

[5] https://pdfbox.apache.org/

[6] https://github.com/google/gson

[7] https://github.com/TomasMikula/RichTextFX

5.3 Issues

Two main issues occurred during the implementation of the system: the creation of exact dates from named entity dates, and the creation of an encapsulated timeline. A minor issue is also explained, which is based on the input documents being grammatically incorrect.

5.3.1 Named Entity Recognition (NER) of Dates

The Stanford CoreNLP tool, allows the resolution of temporal expressions. To explain this, an example is presented. The Stanford tool allows for reference dates to be used when a document is processed. When the tool tags a temporal expression as a DATE, it attempts to normalize it. Note 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[8] format. As can be seen from the format, exact dates are 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. Note that the Stanford normalized dates appear in the format of: yyyy-MM-dd. If BC dates are used, at the start of the normalized result, a '-' is added (indicating a minus date). Using the ISO standard, an algorithm was written to process these dates. Note that, in addition to the possible dates given by the ISO standard, the Stanford tool produces 3 additional possible normalization outputs.

Normalizations refer to the attempt to produce a date from a temporal expression. It may be that, even with a reference point, the tool does not produce an exact date. 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 is that these normalizations do not allow for the comparison of events, which is required to sort them (as the start and end dates are not comparable). To allow for comparison, the most general dates for these references are derived. Since a "PRESENT_REF" refers to

[8] https://www.cl.cam.ac.uk/~mgk25/iso-time.html

the present moment, it can be deducted that it represents the moment in which the text was written in, as that is the time context in which the author wrote it. Thereby, the decision was made to produce as a general date for "PRESENT_REF" a date that is the reference point provided by the user. As the reference point is supposed to be the date in which the text was written in. Note that the user can change the reference point to when-ever they would like, not just the assumed written date of the document.

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 is used. This is because it can be determined that an ambiguous mention of the past would fall anywhere within that period, however an exact determination cannot be made due to the temporal expression not being 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. This is because it would be appropriate for a mention of the future to refer to a moment between now (i.e. the present moment in which the text is presumed to be written in), and until the end of times. However, as a limitation to the system, the end of times is considered the date 31-12-9999. A snippet of the implementation of the algorithm is presented in Figure 5.1.

Note that even though the Stanford CoreNLP library is well documented, it was difficult to find all the different outputs for the normalization of NER dates. It was after the discovery of Stanford's TimeML usage that a document describing the possible outputs of normalized NER dates was found[9].

[9] http://www.timeml.org/timeMLdocs/TimeML.xsd

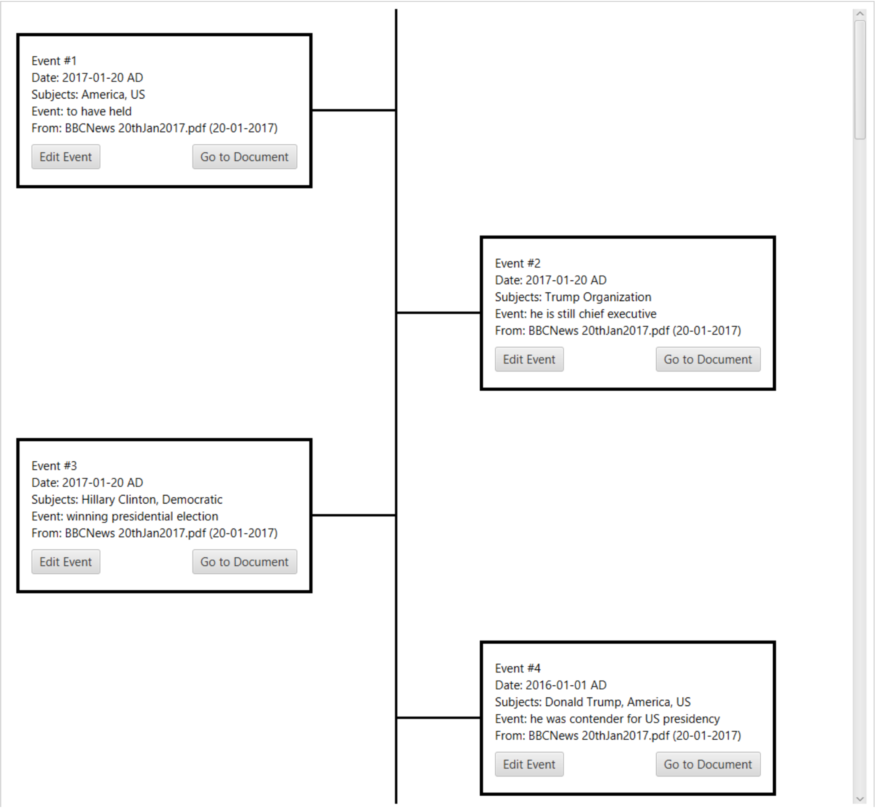
```
private ArrayList<Date> getDate(String date) {
   ...
   /* Set up variables for processing */
   if (onlyPastRefPattern.matcher(date).matches()) {
           //past so make range from 0001-01-01 -> base date (range)
           if (yearMonthDayPattern.matcher(baseDate).matches()) {
               //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 5.1: Part of the Implementation of Resolution of Normalized NER Dates

5.3.2 Encapsulated Timeline View

The initial representation of the events was a traditional timeline (see Figure 5.2). 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 period, they still appear one after the other. The issue is that unless the user specifically looks at the dates associated to the events, it is not possible for them to determine that the events occurred in the same period, instead they may believe that one occurred after the other. This clearly violates the visibility objective 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, and another view where the dates encapsulate the events. For example, if there is more than one event that occurs on the "25-01-2017", then instead of listing them both, one after the other, they are grouped visually by their date. However, visually grouping the events may lead to a "bin-packing" problem. The "bin-packing" problem consists of a set of bins, in this case a set of graphical views, which need to be fit in a finite space, in this case a box of fixed width and height. To deal with this issue, scrollbars can be used in the containers that hold the event layouts. Scrollbars allow for a container to be of infinite height (and/or width). Thus, being able to place the bins (graphical representations of events) in the container without considering the space limit. This view has been given the name "Range View". It 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 algorithm was discussed in the Design Chapter.

Figure 5.2: Screenshot of the Traditional View of the Timeline of Events



Event #1
Date: 2017-01-20 AD
Subjects: America, US
Event: to have held
From: BBCNews 20thJan2017.pdf (20-01-2017)

Event #2
Date: 2017-01-20 AD
Subjects: Trump Organization
Event: he is still chief executive
From: BBCNews 20thJan2017.pdf (20-01-2017)

Event #3
Date: 2017-01-20 AD
Subjects: Hillary Clinton, Democratic
Event: winning presidential election
From: BBCNews 20thJan2017.pdf (20-01-2017)

Event #4
Date: 2016-01-01 AD
Subjects: Donald Trump, America, US
Event: he was contender for US presidency
From: BBCNews 20thJan2017.pdf (20-01-2017)

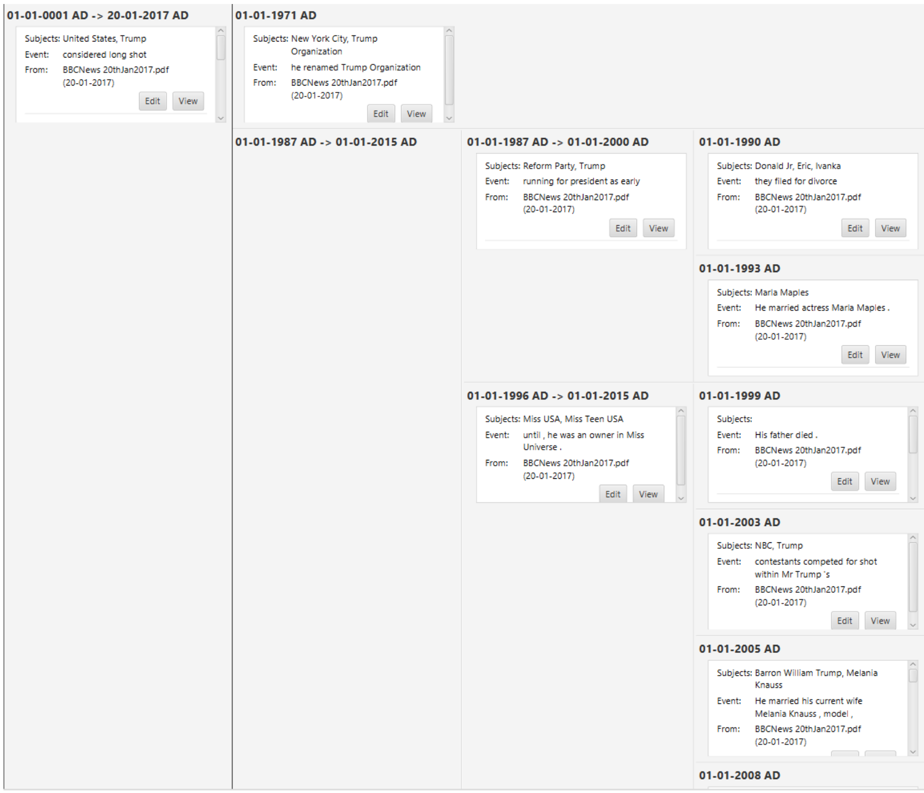
The algorithm behind the Range View, is having a list of Ranges, where each Range is a root node of a Tree of Ranges. Each Range may hold zero or more Results (i.e. events), and a set of children Ranges (zero or more). When the timeline needs to be produced, the list of Range roots is iterated over. The list of Ranges must have been sorted by their start date first. For each a GridPane is made. A GridPane is a layout which consists of rows and columns that can contain sub-views (where a sub-view is a view, i.e. a graphical component). In the first column, 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 are recursively set. The algorithm is presented in Algorithm 5. It is carried out for each Range root in the forest of Ranges.

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

```
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
GridPane toReturn;
toReturn set the number of rows to the size of the input;
toReturn set the number of columns := 2;
for $i$ := $0 \to n$ do
    Range := input list at $i$;
    set up layout for this Range, and set it in toReturn at position $(i, 0)$;
    set its column span at $(i, 0)$ to remaining;
    get layout for the children of this Range := getRangeLayout(range.children);
    set this layout in toReturn at position $(i, 1)$;
end
return toReturn;
```

In the implementation, it was decided to include 3 columns, to have a separator between a Range and its children. Thereby, improving the visibility of the timeline, as the user can differentiate between the Results of a range of dates, and the Results of a sub-range. The individual layout of a Range is listing the Results it holds, and the start and end date for this Range. An example look of the Range View can be found in Figure 5.3.

Figure 5.3: Screenshot of the Range View of the Timeline of Events



01-01-0001 AD -> 20-01-2017 AD

Subjects: United States, Trump
Event: considered long shot
From: BBCNews 20thJan2017.pdf (20-01-2017)

01-01-1971 AD

Subjects: New York City, Trump Organization
Event: he renamed Trump Organization
From: BBCNews 20thJan2017.pdf (20-01-2017)

01-01-1987 AD -> 01-01-2015 AD

01-01-1987 AD -> 01-01-2000 AD

Subjects: Reform Party, Trump
Event: running for president as early
From: BBCNews 20thJan2017.pdf (20-01-2017)

01-01-1990 AD

Subjects: Donald Jr, Eric, Ivanka
Event: they filed for divorce
From: BBCNews 20thJan2017.pdf (20-01-2017)

01-01-1993 AD

Subjects: Marla Maples
Event: He married actress Marla Maples .
From: BBCNews 20thJan2017.pdf (20-01-2017)

01-01-1996 AD -> 01-01-2015 AD

Subjects: Miss USA, Miss Teen USA
Event: until , he was an owner in Miss Universe .
From: BBCNews 20thJan2017.pdf (20-01-2017)

01-01-1999 AD

Subjects:
Event: His father died .
From: BBCNews 20thJan2017.pdf (20-01-2017)

01-01-2003 AD

Subjects: NBC, Trump
Event: contestants competed for shot within Mr Trump 's
From: BBCNews 20thJan2017.pdf (20-01-2017)

01-01-2005 AD

Subjects: Barron William Trump, Melania Knauss
Event: He married his current wife Melania Knauss , model ,
From: BBCNews 20thJan2017.pdf (20-01-2017)

01-01-2008 AD

5.3.3 Incorrect Input Documents

An important issue relating to the input documents is their format and grammar. NLP tools such as StanfordCoreNLP require certain constraints to be applied onto the input documents to be able to annotate them. These tools apply grammatical rules, thereby it is important that documents are grammatically correct to be able to correctly annotate them. This was discussed in the Background Chapter with a discussion on NLP tools being applied to Tweets. For this implementation, the documents are required to be in English as only the English models are loaded for the Stanford tool. The tool supports other languages[10] (such as German, and Spanish), however these require their own models. In addition, the algorithm applied to produce the summaries only works on English written text, as it uses the English grammar.

An issue discovered, was that sentences that are separated by a period, but are not followed by a space, are treated as a single sentence. Thereby, it may be possible that only half the events are detected, if every two sentences are only separated by a period (and not a space). In addition, the summary would be applied to the 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

[10] http://stanfordnlp.github.io/CoreNLP/human-languages.html

date, 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 minimum or maximum is found).

This issue cannot be prevented, as it would require manipulation of the input documents, and it can be the case that the user does not want the system to manipulate the input, but rather process it. Hence, it is advised to users to ensure the documents are in grammatically correct English, for the best possible results. Since the system will be used by law professionals, that are handling formal documents, it can be assumed that these documents would follow this format.

5.4 Testing

Unit Tests were the testing focus in this project. A Unit Test is when individual units (or pairs) of source code of a system are tested to determine whether they are working correctly [1]. As the system was implemented in Java, the library used to aid this is JUnit[11]. Unit tests are primarily done on the back-end of a system, as the focus is on correctness in the logic, not in the UI. Automated tests carried out on the UI are called Instrumentation Tests. They involve emulating users' interaction with the system, i.e. pressing a button or filling in a text field. Both Unit and Instrumentation tests are 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 its graphical representation. The representation is used to display the results; however, the system could also be used to process texts. Thus, the resulting intermediate JSON would be used in a third-party visual representation. Therefore, Instrumentation tests were not carried out.

A total of 35 tests were developed. The main advantage of these, is that when new features are developed, tests can be ran to ensure the rest of the system functions as expected. If the test cases are extensive and appropriate, then it can be assumed that the system will work correctly with the new features. In addition, this is aided using Git (in Git flow), where the features are developed on separate branches, and are only merged to the current working version of code if all the tests pass.

The focus was on the Engine (the component that takes as input, text, and produces lists

[11] http://junit.org/junit4/

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 (a timeline). Tests perform assertions. Assertions are when an actual output of code is checked with an expected output.

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, to produce Ranges, the complex test case expects multiple Range trees of different heights to be produced. Due to the benefit of Gradle, the tests can be ran, after building the project, using the command:

gradlew test.

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.

5.5 UI

From the wireframes presented in the Design Chapter, the actual User Interface(UI) was developed. The screenshots of the different windows are presented in the Figures below (see Figures 5.4, 5.2, 5.3, 5.5, and 5.6). Note that the interfaces provide the requested functionality through buttons and menus. Colour is missing as the focus was on functionality, visibility, and simplicity, instead of the UI being aesthetically pleasing. The system is aimed to be used for task completion, not enjoyment.

Figure 5.4: Screenshot of the Start Up View

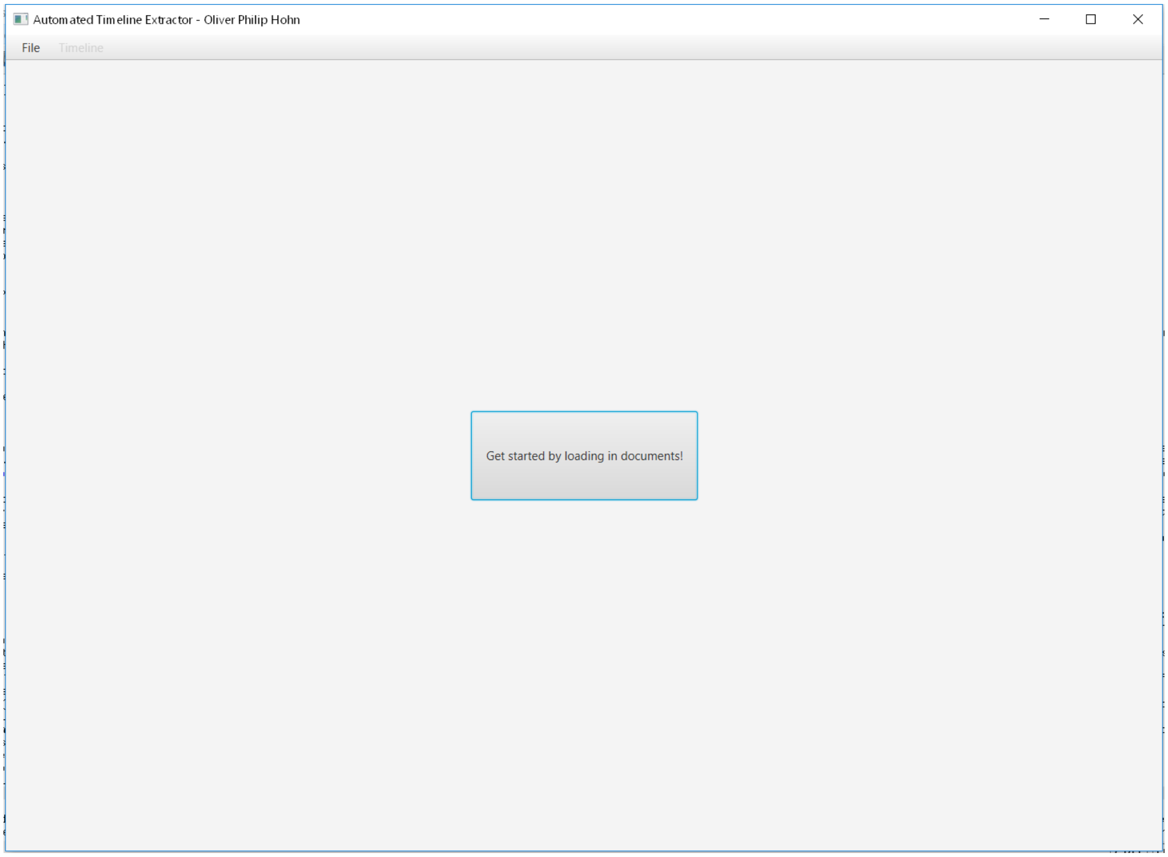
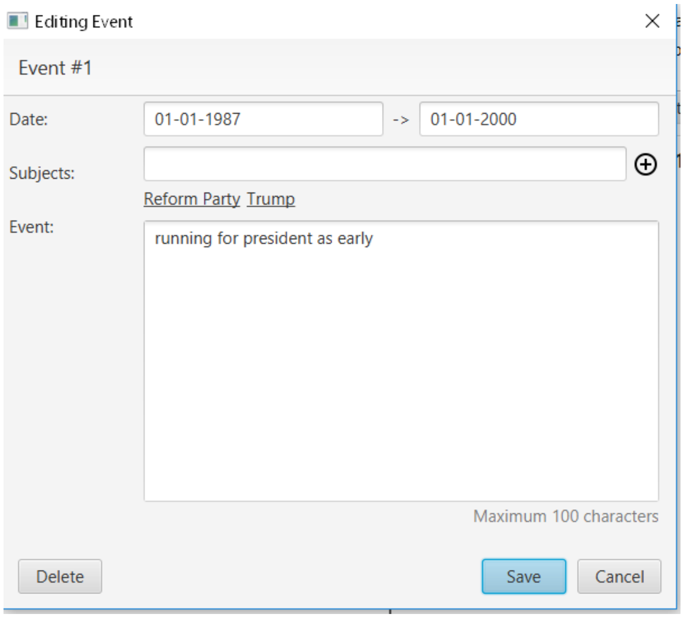

Automated Timeline Extractor - Oliver Philip Hohn
File
Timeline
Get started by loading in documents!

Figure 5.5: Screenshot of the Edit Dialog View

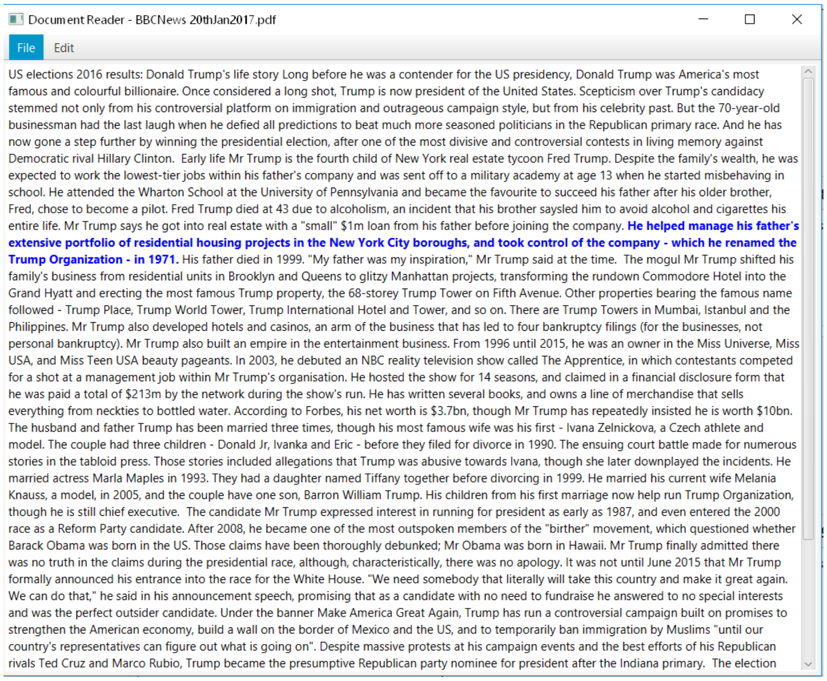


Date: 01-01-1987 -> 01-01-2000

Subjects: Reform Party Trump

Event: running for president as early

Figure 5.6: Screenshot of the Document Viewer



US elections 2016 results: Donald Trump's life story Long before he was a contender for the US presidency, Donald Trump was America's most famous and colourful billionaire. Once considered a long shot, Trump is now president of the United States. Scepticism over Trump's candidacy stemmed not only from his controversial platform on immigration and outrageous campaign style, but from his celebrity past. But the 70-year-old businessman had the last laugh when he defied all predictions to beat much more seasoned politicians in the Republican primary race. And he has now gone a step further by winning the presidential election, after one of the most divisive and controversial contests in living memory against Democratic rival Hillary Clinton. Early life Mr Trump is the fourth child of New York real estate tycoon Fred Trump. Despite the family's wealth, he was expected to work the lowest-tier jobs within his father's company and was sent off to a military academy at age 13 when he started misbehaving in school. He attended the Wharton School at the University of Pennsylvania and became the favourite to succeed his father after his older brother, Fred, chose to become a pilot. Fred Trump died at 43 due to alcoholism, an incident that his brother saysled him to avoid alcohol and cigarettes his entire life. Mr Trump says he got into real estate with a "small" $1m loan from his father before joining the company. **He helped manage his father's extensive portfolio of residential housing projects in the New York City boroughs, and took control of the company - which he renamed the Trump Organization - in 1971.** His father died in 1999. "My father was my inspiration," Mr Trump said at the time. The mogul Mr Trump shifted his family's business from residential units in Brooklyn and Queens to glitzy Manhattan projects, transforming the rundown Commodore Hotel into the Grand Hyatt and erecting the most famous Trump property, the 68-storey Trump Tower on Fifth Avenue. Other properties bearing the famous name followed - Trump Place, Trump World Tower, Trump International Hotel and Tower, and so on. There are Trump Towers in Mumbai, Istanbul and the Philippines. Mr Trump also developed hotels and casinos, an arm of the business that has led to four bankruptcy filings (for the businesses, not personal bankruptcy). Mr Trump also built an empire in the entertainment business. From 1996 until 2015, he was an owner in the Miss Universe, Miss USA, and Miss Teen USA beauty pageants. In 2003, he debuted an NBC reality television show called The Apprentice, in which contestants competed for a shot at a management job within Mr Trump's organisation. He hosted the show for 14 seasons, and claimed in a financial disclosure form that he was paid a total of $213m by the network during the show's run. He has written several books, and owns a line of merchandise that sells everything from neckties to bottled water. According to Forbes, his net worth is $3.7bn, though Mr Trump has repeatedly insisted he is worth $10bn. The husband and father Trump has been married three times, though his most famous wife was his first - Ivana Zelnickova, a Czech athlete and model. The couple had three children - Donald Jr, Ivanka and Eric - before they filed for divorce in 1990. The ensuing court battle made for numerous stories in the tabloid press. Those stories included allegations that Trump was abusive towards Ivana, though she later downplayed the incidents. He married actress Marla Maples in 1993. They had a daughter named Tiffany together before divorcing in 1999. He married his current wife Melania Knauss, a model, in 2005, and the couple have one son, Barron William Trump. His children from his first marriage now help run Trump Organization, though he is still chief executive. The candidate Mr Trump expressed interest in running for president as early as 1987, and even entered the 2000 race as a Reform Party candidate. After 2008, he became one of the most outspoken members of the "birther" movement, which questioned whether Barack Obama was born in the US. Those claims have been thoroughly debunked; Mr Obama was born in Hawaii. Mr Trump finally admitted there was no truth in the claims during the presidential race, although, characteristically, there was no apology. It was not until June 2015 that Mr Trump formally announced his entrance into the race for the White House. "We need somebody that literally will take this country and make it great again. We can do that," he said in his announcement speech, promising that as a candidate with no need to fundraise he answered to no special interests and was the perfect outsider candidate. Under the banner Make America Great Again, Trump has run a controversial campaign built on promises to strengthen the American economy, build a wall on the border of Mexico and the US, and to temporarily ban immigration by Muslims "until our country's representatives can figure out what is going on". Despite massive protests at his campaign events and the best efforts of his Republican rivals Ted Cruz and Marco Rubio, Trump became the presumptive Republican party nominee for president after the Indiana primary. The election

Note that the Range view (see Figure 5.3) is zoomable (i.e. can zoom in/out). This allows the user to have a broader image of the Range View, if it is required. Therefore, allowing the user to obtain a general picture of what occurred.

The data presented of an event includes its subjects (key words) and summary. As the events are encapsulated in Ranges, it is sufficient to show the date(s) for the Range at the top. This demonstrates to the user that the following events occurred during that period. This separates meta data (e.g. when an event occurred), from the actual event, which is what occurred in the event. Each event is accompanied by two buttons, the operations are: to edit the event (which includes deleting it, as can be seen from Figure 5.5), and viewing its document.

An advantage of the Traditional View over the Range view, is memory efficiency. Since the Range view has the function of being zoomable, due to programming language constraints, it cannot be implemented through a ListView. A ListView is a layout data structure where objects of a list are placed row by row. The ListView is memory efficient. It only produces the layout for a row when it needs to be shown. For example, for a list of 10000 items, it is very expensive to hold in memory the individual layouts of 10000 rows, especially if only 5 are being shown. Instead, only the 5 that are being shown are held in memory, and the rest are generated when needed. This is not the case with the Range View. 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. For large sets of events, the traditional view is recommended.

5.6 Important Algorithms

This section presents note-worthy algorithms, in specific their implementation.

5.6.1 Processing Files

The pseudo-code of processing files was presented in the Design Chapter. The use of semaphores was also mentioned. Semaphores enforce that only n processes can acquire their lock, with the $n+1$ process having to wait until another process releases its lock. However, the implementation is not trivial.

The system must allow a certain maximum number of threads to be ran in parallel to process documents. Processes must not fall in deadlock, which is when the processes are indefinitely allowing other processes to run, such that no run, or starve, which is when a process waits indefinitely to run. This can occur when semaphores are used, and they are not released when a process finishes its task. To ensure the semaphores are released, a call-back is given to the threads, which is to be used at the end of the threads parallel processing. If an error occurs, the semaphore is still released, allowing the next thread to run. Multi-threading precautions must be considered. When the call-back 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 a Result list at the same time. Fortunately, Java provides the **synchronized** keyword for methods (see Figure 5.7). 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.

```
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 5.7: Implementation of call-back after Files have been processed

Threading precautions have been taken throughout the project. Where threads process sets of data, it was ensured that the data items in the set can be processed independently of each other. This cannot be possible in all situations, in such cases synchronization and semaphores are used to ensure concurrency.

5.6.2 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 if a Result can be added to an existing Range. When a Result is attempted to be added to a pre-existing Range, an **add()** (see Figure 5.8) method is called. In this method, it is initially checked whether the Result should be added to this Range. This check involves looking at the dates of the Result and checking whether the Range's dates fully or partially encapsulate them. If this is not the case, then the attempt of adding the Result fails. If the Range encapsulates the Result's dates then it will be checked whether the Result can be added to any of the existing nodes of the Range tree. This would occur if there is any node in the tree that holds the same exact dates as the Result. If this is the case then the Result will be added to that node. If this is not the case, then it can be determined that the Result belongs to this tree, however its place in the tree needs to be created. It can be the case that there is a Range that partially encapsulates the Result, if so the Range would need to be expanded. In both cases, the tree needs to be traversed to find the optimal position for the Result.

```java
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 5.8: Implementation of Adding Results to a Range

5.6.3 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, the file that is saved, is a graphical representation of the events, using the traditional timeline view. It is aided by the Apache PDFBox and Commons library. The libraries work like a painting tool. The pages are canvases, where lines and text can be drawn 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, certain constraints are required. For example, the maximum number of events to be displayed on each page, and the size of the text of the summaries and subjects. Due to tool limits, it is not possible to continue drawing on the next page of the PDF document, hence events cannot overflow onto the next page. However, since the text that is used in the summary should be short, assumptions can be made. For example, the maximum size of a container that holds an event. Knowing this, allows the drawing of containers of maximum size on the document. Thus, ensuring that all the data for each event will fit in a container. Therefore, the system can be considered as drawing containers on the page and filling them with their event data. This is repeated for all the events, and where necessary creating a new PDF page to fill more events. Hence the task, has been broken down to moving the x and y positions, writing text (the event data), and drawing the rectangle (container). Since in the traditional view, the layout of the events is one on the left of the page and the other on the right of the page, the tasks must be changed minimally to allow for this. In Figure 5.9, the implementation, to draw an event on the right of the page, is provided.

```
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 5.9: Implementation of drawing for a Result in the PDF timeline

The JSON implementation, is aided by the Google GSON library. The JSON format, provides a clearly defined key-value data format that can be interpreted by any system. The advantage of this library is that it allows for a flexible creation of JSONs. Specifically, it allows the developer to specify how an object of a given type should be serialized (see Figure 5.11), i.e. how its data should be extracted into JSON format. In this case, the objects to be serialized are Result objects. The data extracted from a Result includes its start and end date, its list of subjects, its summary, and its file data. This is done through an adapter. The adapter defines how the data of the Result object is extracted, and how a JSON Object is created. For a list of Results, this will be carried out for each Result, and thereby, a resulting list of JsonObjects will be produced, i.e. a JsonArray. This can then be saved by the user in a file, and can then be interpreted by any third-party system. The format of a JSON of a Result is given in Figure 5.10. Where G, in the dates, is the ERA (i.e. BC or AD) of the date.

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

Figure 5.10: Structure of Result (event) JSON

```
public JsonElement serialize(Result src, Type typeOfSrc, JsonSerializationContext context)
    {
   //json object for this result
     JsonObject jsonObject = new JsonObject();
   //adding the range dates (which can be null)
   /* whenever a value is null, the key-pair will not be included in the final JSON */
   jsonObject.addProperty("date1", src.getTimelineDate().getDate1FormattedDayMonthYear());
   jsonObject.addProperty("date2", src.getTimelineDate().getDate2FormattedDayMonthYear());
   //adding the subjects
   JsonArray subjectJsonArray = new JsonArray();
   for (String subject : src.getSubjects()) {
     subjectJsonArray.add(subject);
   }
   jsonObject.add("subjects", subjectJsonArray);
   //adding the event
   jsonObject.addProperty("event", src.getEvent());
   /*adding the file data (excluding the path, since this can be used on other system
       where files are elsewhere)*/
   JsonObject fromJsonObject = new JsonObject();
   FileData fileData = src.getFileData();
   if (fileData != null) {
     fromJsonObject.addProperty("filename", fileData.getFileName());
     fromJsonObject.addProperty("baseDate",
         fileData.getCreationDateFormattedDayMonthYear());
   }
   jsonObject.add("from", fromJsonObject);
   return jsonObject;
}
```

Figure 5.11: Implementation of the Serializer used in the Adapter to generate JSON for a list of Results

Chapter 6

Professional and Ethical Issues

During the development and evaluation of the system, great care has been taken to follow the Code of Conduct[1] and Code of Good Practice[2] issued by the British Computer Society. This must be done to avoid serious legal and ethical problems.

Importance has been given to state explicitly which software libraries and academic papers have been used throughout the development. The information and services of these have been used and modified to meet the requirements of the project. All the software produced consists of my own work, except where it is explicitly said otherwise.

As the system is intended to be released as open-source, it has been confirmed that the libraries allow this. If the system were to be used commercially, a license would have to be produced that would encapsulate the project and its libraries.

The developed system does not collect user data, thereby abiding to the public interest of the Code of Conduct. In addition, for the evaluation phase it was checked whether ethical approval would be required. It is not, since the test participants are kept completely anonymous, and the data sources used are publicly available.

It should be noted that if the system is moved onto a server to produce timelines from large sets of documents, the processing power used would increase substantially, and may affect other systems running on the same machine. This can be avoided by modifying the settings of the system, e.g. the number of threads used, to reduce the number of processes running.

[1] http://www.bcs.org/category/6030

[2] http://www.bcs.org/upload/pdf/cop.pdf

Chapter 7

Evaluation

As presented in the Design chapter, the objectives of the system are: visibility, efficiency, and effectiveness. To evaluate each of these, the chapter is split into three sections. Each addressing an objective.

7.1 Visibility

To evaluate the visibility of the system, an expert heuristic using the Nielsen Usability Heuristic [14] set was used. This involves an expert evaluating the UI of the system with a set of principles, which have been tested, and then discussing the system for each principle. The evaluation can be formal or informal. The former was chosen, as it was carried out at the end of the development. Informal evaluations are done when analysis needs to be done during the iterations of design, since it provides feedback quicker and allows designers to discuss the problems when producing the next design. Formal evaluation is done, usually, at the end of the development cycle when the system is being evaluated. In this case, the focus is on evaluating visibility. Heuristics do not only focus on visibility, since some of the principles focus on error prevention. However, this is also important for a system that is to be simple to use.

Other heuristic sets exist; however, the Nielsen set is the most known[1]. Another prominent design evaluation is cognitive walkthroughs. They consist of giving experts, scenarios to go through, and during their progression of the scenarios, they answer three questions related to design issues [19].

Due to time constraint, only one in-depth expert heuristic could be carried out. Normally, there would be multiple experts, each carrying out their own evaluation, and then they would

[1] https://www.usability.gov/how-to-and-tools/methods/heuristic-evaluation.html

discuss between themselves to produce a single document detailing the usability problems. In this case, the expert has experience in applying the principles from previous projects. Note that they do have technical knowledge in Computer Science, hence they are not considering the system from a novice users perspective, but rather in identifying where the system satisfies the principles. Experts may be biased on how they believe a design should be, however this is expected as designs are a subjective matter.

The data collected from an expert review is qualitative data. Qualitative data are statements, observations, and subjective judgements. In this case, they are issues in the system with possible solutions, which aim to satisfy the principles they violate. It is different to the quantitative (numerical) data collected in the Effectiveness section.

The main principles evaluated are visibility, match with the real world, consistency, and error prevention. With a focus on visibility.

For the visibility principle, where the focus is on the user knowing the systems state, the design succeeds according to the expert. The user is aware of the actions available to them as the system progresses through its states of processing files. This begins from the system only allowing the user to load documents on start-up, to the loading dialog when the documents are being processed, and to the subsequent viewing of the produced timeline. To ensure error prevention, where data validation is done on user input, red-borders are shown in the input fields when the data is invalid and the options to save the data are disabled. This allows the user to determine that a mistake was made with the input, and specifies where the mistake occurred. To ensure the user knows what the expected input format is, the input fields are aided with hints. For example, the format of dates is shown when the user must enter the base date for a document, or a character limit is shown when the user is editing the summary of an event. Clearly marked exit, "Cancel" buttons, are provided to allow the user to not commit their changes. Confirmation dialogs are also used to prompt the user to confirm an action that may be costly, e.g. deleting the subjects of an event, or deleting the event itself. All the previously mentioned design choices, accompanied with the non-technical language used in the system, ensure that the system should be accessible to any user.

To improve visibility, the expert suggested improvements. For example, hinting the user on how to delete subjects, or providing an indicator for the zooming in/put function in the "Range View". To aid the user recognition of the function of buttons, it was suggested to use icons that visually describe the function. The expert suggested filling up a progress bar, instead of the circular progress bar in the system, when documents are processed. However, this is not a clear indication of the progress as some documents require more time to process than others, but in the progress bar they would have equal impact.

From this evaluation, it can be determined that the system fulfilled its aim of producing a system that is visible to the user. The user is informed of the states the system is in as it processes documents. Clearly marked exists, and error prevention with indicators of what the error is, are provided. Note that visibility is not as important as the other objectives, as the system could be used as a library (directly, or indirectly through JSON outputs) where another developer produces their own UI. However, in the case that the system is used by a user (not a developer), the UI produced should be accessible for them, and provide useful information.

7.2 Efficiency

Efficiency relates to the time it takes for the system to produce an answer. A system is efficient if it produces a result in an appropriate amount of time based on its input.

A way to evaluate this is to run the system on different inputs and record the time taken to produce a timeline. However, the time recorded is specific not only to the machine that the system is being run on, but also how the CPU scheduled the work at that point in time (i.e. how much other work was being carried out on the machine). If the system is running with other tasks running in the background and the CPU gives the application less priority, i.e. other tasks run before this one, then the system will require more time to produce a result. Note that depending on the hardware's computation-power, when running the system, results may be produced quicker or slower. In addition, not all input sizes can be tested, as the document set size can be infinite. The solution to this is to consider the time complexity of the main algorithms used in the system.

The time complexity of an algorithm is, the number of operations, and time required to produce an output for an input of a given size [17]. This allows to determine how well the system scales with larger inputs, by only considering how many operations are performed as a function of the input.

Time complexity focuses on the worst-case scenario, and does not consider low-order operations. For example, if for an input n an algorithm performs $3n + 2$ operations, the low-order term 2 and the co-efficient 3 are omitted. This would result in a time complexity $O(n)$ (where O is called big-Oh). In general, the highest order-term is taken, and all the rest are omitted. For example, $4n^2 + 10n + 2$ would have time complexity $O(n^2)$. Low-order terms are omitted since with a very large input, or as $n \to \infty$ (n tends to infinity), the time required for the low-order operations becomes irrelevant. The co-efficient of the high-order term is omitted, as a machine can be 4 times faster, or 4 times slower, so it does not affect, in general, the number of operations performed.

To measure the efficiency of the system, the time complexity of the front-end (graphical part) and back-end (logical part) of the system have been computed separately, as the system can be used standalone, or as a library in other systems that produce their own UI. The complexity of each algorithm is presented in the table below (see Figure 7.1).

Algorithm	Information	Time Complexity
Document Processing	n - number of documents s - number of sentences w - number of words	$O(nsw^2)$
Range Production	n - number of Results	$O(n^2)$
Range Timeline View	n - number of Results	$O(n)$
Traditional Timeline View	n - number of Results	$O(n)$

Figure 7.1: Time complexity of main Algorithms in System

7.2.1 Back-End

Processing Documents

The two main algorithms are processing of documents, and the production of Ranges. The complexity of each is presented and discussed. It is assumed that the time complexity of the Stanford CoreNLP tool to annotate a document is $O(w)$ (for all w words in the document), and that the documents are annotated before being processed.

The algorithm used was previously presented (see Algorithm 3). For a list of n documents, each is processed. The algorithm can process at most x documents in parallel at any time. Depending on the user's settings value x can be 1 or ∞ (infinity). If ∞, then it can be determined that the complexity of the algorithm is given by the complexity of processing the largest document, since all documents are being processed in parallel.

When a document is processed, each sentence is checked for a temporal expression, before a summary or any other processing is done. In the worst case, all sentences s in a document must be fully processed. In such a case, the dates, the subjects, and the summary need to be produced. These are all done after each other, so it can be determined that the computation complexity of processing one sentence is given by $max(getDate, getSubjects, getSummary)$. Where max will return the greatest time complexity of the three operations.

To get the date of a sentence, with w words, each temporal expression needs to be processed.

The processing of a temporal expression is linear (see the getDate implementation in Figure 5.1), as it does not depend on the input directly. It performs at most 3 loops to produce an exact date. Thus, processing an NER date has a complexity of $O(1)$. In the worst case, every word in the sentence is a temporal expression (which does not happen normally in a sentence of well-written documents, but can still occur). Thereby, the time complexity for getDate is $O(w)$.

To select the subjects of a sentence, the NER annotator is used to determine which words are of interest. In the worst case, all words can be of interest. When a word is identified, it is placed in a list, for the subjects of that event. Placing a word in a list has time complexity $O(1)$, hence it being done for w words, in a sentence, leads to a complexity of $O(w)$.

To create the summary of a sentence, with w words, the hedge-trimmer algorithm is used (see the Algorithm presented in the Design Chapter). In the paper, the time complexity of the algorithm is not presented as the algorithm is not explicitly provided, but rather explained. Thereby, the algorithm that was implemented is analysed. In the grammatical tree produced, each word is a leaf. The structure of the tree varies, however for this evaluation it is assumed that in the worst case a full-binary tree is produced. This may not be the case, as the trees produced vary in format, however it is required to proceed with the evaluation. If there are w leaves in the tree (i.e. each word in the sentence is a leaf), then there are $2w - 1$ nodes in the tree. The rules of the algorithm are applied one after the other. Thereby, the time complexity of the algorithm is given by the rule with the highest time complexity. The first two rules traverse the tree once, and have a complexity of $O(2w - 1) = O(w)$. In the last step, the tree is iteratively shortened until it is below the threshold. In the worst case, the threshold can be 0 (not explicitly allowed in the system as the minimum value is 10). Note that the threshold value is met when the number of words in the summary falls below it, and that the size of the summary is given by the number of leaves in the current grammatical tree. At each step, at least one node needs to be removed for the algorithm to continue until the threshold is met. Note that if an inner node is removed, its children are also removed. Thus, the leaves are reduced, thereby getting closer to the threshold value. To identify the node to remove, the tree must be traversed, and as presented earlier, this has complexity $O(w)$. In the worst case, only one leaf is removed each time. Thus, the tree is traversed w times. Therefore, the time complexity of the last rule is given by $O(w^2)$.

Therefore, the processing of a sentence with w words is given by $O(w)+O(w)+O(w^2)$, where each time complexity corresponds to a task (inferring dates, getting subjects, and producing a summary). Thereby, the time complexity of processing a sentence is $O(w^2)$, as the low order terms are removed.

Assuming the set of n documents is processed one after the other (i.e. at most 1 document can be produced in parallel, in the worst case), and that all the documents have s sentences (or less). Where each sentence has w words (or less). The running time of processing one document is $O(sw^2)$. Thus, the running time of processing n documents is $O(nsw^2)$. Where it is not known which of n, s, or w is the high-order term. The time complexity of annotating a document is omitted as it can be considered that $sw^2 > w$.

If the input consists of many documents, with few short sentences (i.e. w and s are smaller than n), then the running time is given by $O(n)$. Which for processing n documents, is efficient since the system scales linearly with the input. However, this is not always the case, as it may be that there are many short sentences (i.e. s is greater than n and w, thus the time complexity is $O(s)$), or there are few long sentences (i.e. w is greater than n and s, thus the time complexity is $O(w^2)$).

Range Production

Ranges are produced with the algorithm presented in the Design Chapter. Based on an input of n Results, they must be sorted, added to an existing Range tree or creating a new Range trees, and then, finally, the trees must be sorted.

Sorting Results in Java has a complexity of $O(n \log n)$. This is due to Java using merge-sort when comparing the results. Merge-sort consist of recursively breaking down the problem space in half (which produces the $\log n$ part), and then building the sequence back up (which produces the n). However, it should be noted that with sequences that are almost sorted in Java 8+, due to the use of TimSort[2], the time complexity is closer to $O(n)$.

Each Result needs to be added to a tree. Before the tree is traversed to find a location, a check is performed to determine whether it needs to be added to the tree. In the worst case, the Result cannot be added to any tree, thus all the trees are checked and a new Range tree needs to be made to hold the Result. This can be the case when the dates of events are disjoint. In such cases, the number of Trees that need to be checked increase by 1 for each Result added. Assuming the time complexity to check whether a Result can be added to a tree is $O(1)$ (since the check is in constant time), then the time complexity is given by the sum of 0 to n. As in the first step, no Ranges exist so nothing needs to be checked. Then one tree needs to be checked,

[2] https://bugs.openjdk.java.net/browse/JDK-6804124

then two trees, and so on. Where at the nth Result, $n-1$ trees need to be checked, each with a complexity $O(1)$. Thereby, the total complexity is given by the sum of 0 to n, which is given by $n(n+1)/2$. This produces a time complexity $O(n^2+n/2)$, which is $O(n^2)$.

After all the trees have been produced, the Ranges need to be sorted. In the worst case, there is one fully expanded tree. Which is a tree where for each Result, it had to be expanded because the dates overlapped (but where never fully contained within each other). This leads to a full binary tree as when a Range is expanded it has two children set to it: one being a new Range holding the Result being added, and the other being the previous subtree that the Result's dates partially overlapped with. Using the Java sorting algorithm leads to $O(n \log n)$, as the low-order terms are dismissed. If each Result was in a separate tree, then only the Results need to be sorted, of which there tends to be less than the number of Ranges, hence not being the worst case.

Thereby, the time complexity of the Range Production algorithm is given by the operation with the greatest complexity, which is adding Results to the tree. Thus, the time complexity of the algorithm is given by $O(n^2)$.

7.2.2 Front-End

In this section, the focus is on the creation of the range and traditional timeline views.

Range View

As can be seen by the algorithm to produce the Range view (see the Figure 5 in the Implementation Chapter), each Range is considered separately when it is built. In the worst case, there is one Range with a fully expanded tree (i.e. a full binary Range tree). If the tree holds n results, which are held at the leaves, then it has a total of $2n-1$ nodes. For all nodes, it must produce a layout, which includes the production of its children recursively. Therefore, $2n-1$ layouts are produced, so the time complexity is given by $O(n)$. If there is a separate tree for each Result, then only n layouts need to be produced.

There can be performance issues, as this layout embeds layouts within each other and many views are held in memory at a time. However, the focus in this section is to consider the time for the system to produce the UI, not how heavy it is for the system's memory.

Timeline UI

The creation of the timeline view is trivial. For a list of n results, the layout for each row of the ListView is produced. Hence, n rows are produced. This leads to a time complexity of $O(n)$.

Note that the ListView used in the system does not hold all rows in memory at once. The layout of a row is only produced if it needs to be shown, i.e. that part of the ListView is visible to the user. Hence, the time complexity to produce the timeline view is less than $O(n)$, as the system does not show all the Results. Only when the size of the ListView is smaller than the amount of allocated screen space, will the ListView have to produce n rows as all rows can be shown, and thus will have a time complexity of $O(n)$.

7.3 Effectiveness

Effectiveness of a system is how correct it is at producing results, in this case timelines. To determine whether a timeline is correct, experimental testing was done. This allows to determine whether the produced timeline is similar to the ones produced by people.

7.3.1 Testing Preparations

To test the effectiveness of the timelines, test participants were asked to build manual timelines given by a document. These were then compared to the produced timeline by the system for the same document.

Note that there is no one perfect timeline, but instead there can be many good timelines that express the documents. This is demonstrated by the different timelines produced by the test participants for the same document. The aim is to compare the produced timelines to the manual timelines, and see what are the differences and similarities.

Participants were kept anonymous during the entire process, and will be kept anonymous during the presentation of data. The participants come from different academic backgrounds, are of different ages, and have different levels of exposure to the document data sets. A clear limitation to the experiment was the availability of participants. As such, limited data was gathered.

The data sets consisted of publicly available newspaper articles of different domains: politics, criminal cases, and short bibliographies. To ensure that a substantial amount of data could be gathered, articles that were rich in temporal expressions were picked. This allows for more timeline data points to be compared between a manual timeline and an automated produced timeline.

Using publicly available data sets (e.g. newspapers), and keeping the participants anonymous complies with the guidelines of the BCS, and does not require an ethical approval for the experiment.

7.3.2 Testing

Participants were provided a document, and were asked to select events line-by-line, like the system. This is to allow comparison between the two timelines.

There was no time limit for the participants, as the aim was for them to produce correct timelines instead of producing timelines quickly, since the system has a clear advantage in that regard.

The manual events produced by the participants, included the date of the event and a short summary of what occurred. In situations where the dates cannot be determined, participants would link the events to others by saying it occurred before/after another event. Currently, the system cannot establish these links.

The manual and automated produced timelines are found in the Appendix.

7.3.3 Analysis

The comparison of the manual timelines consisted in: the number of events collected, the percentage of matched and unmatched sentences, the percentage of mismatched exact dates, and the similarity in summaries.

The number of events allows to determine what is the expected number of events for a given document. This can be combined with the other data to determine if the timeline produced is too broad in its event generation, or too specific.

The matching and un-matching of sentences, consists of identifying which sentences the users used in their generation of events and which the system used. This allows to evaluate how effective the system is in identifying an event, independently of whether the data of the generated event is correct.

Mismatched dates consist of determining how accurate the system is in producing exact dates for events. It is expected that the system will perform poorly in situations where the date of an event cannot be identified, but only that it occurred before/after another event. For test participants, this is not a problem, as they understand context and can apply their own thinking to determine whether an event occurred before or after another event, even when there is no temporal data to support this. To determine the quantity of mismatched dates, events from the manual and produced timeline are only considered if they originate from the same sentence in the document. Then the dates of the two events are compared.

The similarities in summaries determines how effective the system is in conveying the meaning of the original sentence that produced it. The events are grouped by the sentence that produced them, and then it is checked whether the summary and key words of the event, from the produced timeline, matches the meaning of the summary of the event from the manual timeline. This is generally done by checking that they have the same keywords.

The results of the following are presented in the table below (see Figure 7.2). The name of the articles used are provided, along with their source and their publishing date.

Note that the percentage of events originating from the same sentence focuses on how many events in the manual timeline were also identified in the automated timeline. This allows to determine how many of the events in the manual timeline are not identified by the automated system.

No statistical analysis could be performed to determine with certainty how accurate the system is, as there is not enough data. However, it can be concluded, that from the data gathered, that the system identifies more than 75% of the events in the documents, if the manual timelines identify all events. Events with completely unknown dates (i.e. there are no explicit temporal expressions) are not identified. The summary generation is not optimal, as just above 65% of the automated timelines capture the meaning of the manual timelines. Thus, showing the weakness of the decision-based summary algorithms, and reasons as to why the domain-dependent statistic models are more prominent. An effective point of the system was the production of exact dates from temporal expressions. For the automated inference of dates, most inferred dates match the dates determined by the test participants.

Therefore, the system does not produce perfect summaries that convey the same meaning as the original sentence. However, the system is effective in the production of exact dates from temporal expressions, and performs well in identifying events.

Comparing System to Manual Timelines From...	Participant 1	Participant 2	Participant 3
Difference In collected events (+/-) (19 events collected by system)	+5	-1	-3
Percentage (%) of events originating from the same sentence	66.7	83.3	87.5
Percentage (%) of events not identified	33.3	16.7	12.5
Percentage (%) of mismatched dates	0.0	6.7	0.0
Percentage (%) of similar summaries (matching keywords)	75.0	73.3	78.6
Unknown Dates of Manual Timelines From...	4	2	2

(a) BBC: US elections 2016: Donald Trump Life Story (Published 20th of January 2017)

Comparing System to Manual Timelines From...	Participant 1	Participant 2	Participant 3
Difference In collected events (+/-) (4 events collected by system)	+5	0	+3
Percentage (%) of events originating from the same sentence	44.4	100.0	57.1
Percentage (%) of events not identified	55.6	0.0	42.9
Percentage (%) of mismatched dates	0.0	0.0	0.0
Percentage (%) of similar summaries (matching keywords)	50.0	50.0	50.0
Unknown Dates of Manual Timelines From...	0	0	-1

(b) Guardian: British Toddle Abducted, Police believe (Published 5th of May 2007)

Comparing System to Manual Timelines From...	Participant 1	Participant 2	Participant 3
Difference In collected events (+/-) (6 events collected by system)	+2	+1	+1
Percentage (%) of events originating from the same sentence	75.0	88.5	71.4
Percentage (%) of events not identified	25.0	12.5	28.6
Percentage (%) of mismatched dates (on matched events)	33.3	16.7	0.0
Percentage (%) of similar summaries (matching keywords)	66.7	66.7	80.0
Unknown Dates of Manual Timelines From...	1	0	1

(c) Guardian: Madeleine McCann detectives arrive in Portugal to question 11 suspects (Published 9th of December 2014)

Figure 7.2: Manual vs Automated Timelines Comparison

Chapter 8

Conclusion and Future Work

8.1 Conclusion of Project

In conclusion, the project aimed to build a system that extracts events autonomously from documents, and then produces a timeline with the events. The Stanford CoreNLP tool was the major library used to categories sets of words into predefined categories (NER annotator), and build the grammatical trees (POS annotator). A trimming algorithm was used to produce summaries of sentences through headline generation. The use of multi-threading allowed for parallel processing of documents, and thus faster times to produce the event data and populate the timelines.

Open-source will allow the project to be further developed, and included in other event extraction systems. Since the code produced is fully documented, and the back-end of the system is separate of its graphical counterpart, the system can be integrated as a library in other projects. The intermediate produced JSON allows the Results of the system to be used in third-party applications. Using Gradle, and a produced wrapper, allows for portability to be achieved, as the required libraries are retrieved before run-time, and the system can be ran with one command[1].

Issues occurred, and new requirements appeared during the production of the system. These include the creation of exact dates from normalized dates, wrong input documents, and the encapsulation of events for a new timeline view. These issues were identified and appropriately solved. Of course, as with any project, the initial design was changed, but through Agile methodology this could be tackled and did not hinder the implementation of the system.

[1] gradlew run

Changes include minor alterations in the systems architecture to allow for the independence between the logic of the system and its graphical representation.

The use of running time complexity allowed for the applications efficiency to be evaluated independently of the system it is run on, and on the data size given as input. Using an expert heuristic allowed the produced UI to be evaluated for visibility. Since the heuristic is a standard, it has been thoroughly tested before-hand by others. From the evaluation of the effectiveness of the system, it can be determined that the system is effective in identifying the events in the documents used (that consisted of different domains). However, the summary module of the system requires further work. It is not possible to say with certainty how effective the system is in general, as not enough data is available for such analysis. However, with more time this can be done.

8.2 Future Work

Future works consist of areas of possible development that would improve the overall effectiveness of the system. The areas are described below. Implementing any of these would be an improvement, however the challenge is in being able to combine them all to produce a system that can be used at a commercial level by law professionals, and would be unrivalled.

Neural Net - As mentioned in the Background Chapter, there is a heavy use of Neural Nets for text summarization, as can be seen from the works of [3] and [16]. These are often combined with noisy-channel models that use data sets for statistical computation of summaries. The main benefit of such a system would be to provide better summaries. The reason for them not being used in the project is the large data set required and the amount of computation done to produce a summary of one sentence.

Machine Learning - Machine learning allows an algorithm to produce accurate results by using a data set [6]. In this system, it could be used in the production of events. For example, when a user edits an event it can be assumed that they produced a corrected event. This data can then be used and considered in the creation of other events by the system, to produce more precise ones.

Cloud - A cloud allows a service to be provided independent of its software and hardware [5]. This is done by deploying the system on a remote server, and giving clients an interface to interact with it. Thereby it is not required for the client to have a powerful machine, and it does not affect their systems performance. The downfall is the cost. However, the performance could be improved as specific hardware could be used to improve the efficiency and it could be combined with Machine-Learning and Neural Nets to allow the system to improve in its event identification and production.

Extending the Stanford CoreNLP tool - An issue in the tool used, is that it does not attempt to link ambiguous temporal expressions, i.e. it may be known that an event occurred before another. For example, a person must be born first before they can work (example taken from [13]). In the system when an exact date cannot be given, an ambiguous date is produced. For example, for both born and working it could produce a "PAST_REF". This would then cause the system to assign both events to the same range of dates, i.e. 0001-01-01 up to the reference point used for the document. However, these dates can be made more precise. Since a person must be born first, and be over the age of 16 to work, the range of dates for the work event could be identified more precisely. The problem arises, from the tool and the produced system considering each sentence independently of each other. It would be beneficial if the events were to be linked one after the other, such that it may not be known when someone was born or when they worked, but that the event of them working is after the event of them being born. This would require building on the NLP tool used in the system.

References

[1] Agile Alliance. Unit testing. https://www.agilealliance.org/glossary/unit-test/. Accessed 9 April 2017.

[2] Dave Burke. Use case actors - primary versus secondary. https://blogs.oracle.com/oum/entry/use_case_actors_primary_versus. Accessed 9 April 2017.

[3] 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.

[4] 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.

[5] Oxford English Dictionary. Cloud computing. https://en.oxforddictionaries.com/definition/cloud_computing, . Accessed 9 April 2017.

[6] Oxford English Dictionary. Machine learning. https://en.oxforddictionaries.com/definition/machine_learning, . Accessed 9 April 2017.

[7] 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.

[8] J. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*, pages 363–370, 2005. URL https://nlp.stanford.edu/~manning/papers/gibbscrf3.pdf.

[9] Capital Community College Foundation. Sentence subjects. http://grammar.ccc.commnet.edu/GRAMMAr/subjects.htm. Accessed 4 Dec. 2016.

[10] L. Iwanska and W. Zadrozny. Introduction to the special issue on context in natural language processing. *COMPUTATIONAL INTELLIGENCE*, 13(3):301–308, 1997. doi: 10.1111/0824-7935.00041.

[11] 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`.

[12] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Cambridge University Press, Reading, Massachusetts, 2009.

[13] 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`.

[14] J. Nielsen. 10 usability heuristics for user interface design. 1995. doi: https://www.nngroup.com/articles/ten-usability-heuristics/.

[15] 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`.

[16] 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`.

[17] Michael Sipser. Introduction to the theory of computation. chapter 10, pages 275–276. Cengage Learning, 2012.

[18] Kristina Toutanova, Dan Klein, Christopher Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of HLT-NAACL*, pages 252–259, 2003. URL `https://nlp.stanford.edu/~manning/papers/tagging.pdf`.

[19] C. Wharton, J. Rieman, C. Lewis, and P. Polson. The cognitive walkthrough: A practitioner's guide. *Usability inspections methods*, pages 105–140, 1994. doi: http://www.colorado.edu/ics/sites/default/files/attached-files/93-07.pdf.

[20] Wikipedia. Natural language processing. https://en.wikipedia.org/wiki/Natural_language_processing. Accessed 9 April 2017.