Project Number: **770299**

NewsEye:

A Digital Investigator for Historical Newspapers

Research and Innovation Action
Call H2020-SC-CULT-COOP-2016-2017

D5.7: Personal Research Assistant: Reporter (c) (final)

Due date of deliverable: M45 (31 January 2022)

Actual submission date: 16 January 2022

Start date of project: 1 May 2018 **Duration:** 45 months

Partner organization name in charge of deliverable: UH-CS

Project co-funded by the European Commission within Horizon 2020		
Dissemination Level		
PU	Public	PU
PP	Restricted to other programme participants (including the Commission Services)	-
RE	Restricted to a group specified by the Consortium (including the Commission Services)	-
CO	Confidential, only for members of the Consortium (including the Commission Services)	-

Revision History

Document administrative information		
Project acronym:	NewsEye	
Project number:	770299	
Deliverable number:	D5.7	
Deliverable full title:	Personal Research Assistant: Reporter (c) (final)	
Deliverable short title:	Personal Research Assistant: Reporter (final)	
Document identifier:	NewsEye-T52-D57-Reporter-c-submitted-v6.0	
Lead partner short name:	UH-CS	
Report version:	V6.0	
Report preparation date:	16.01.2022	
Dissemination level:	PU	
Nature:	Report	
Lead author:	Leo Leppänen (UH-CS)	
Co-authors:	Elvys Pontes (ULR), Jose G. Moreno (ULR), Lidia Pivovarova (UH-CS), Hannu Toivonen (UH-CS)	
Internal reviewers:	Max Weidemann (UROS), Guenter Hackl (UIBK-DEA)	
Status:		Draft
		Final
	x	Submitted

The NewsEye Consortium partner responsible for this deliverable has addressed all comments received, making changes as necessary. Changes to this document are detailed in the change log table below.

Change Log

Date	Version	Editor	Summary of changes made
15/01/2021	0.1	L. Leppänen (UH-CS)	Initial version based on D5.4
11/03/2021	1.0	L. Leppänen (UH-CS)	Ready for project-internal review
15/03/2021	1.1	G. Hackl (UIBK-DEA)	Internal review
16/03/2021	1.2	M. Weidemann (UROS)	Internal review
16/03/2021	1.3	H. Toivonen (UH-CS)	WP leader's review
18/03/2021	1.4	L. Leppänen (UH-CS)	Addressed most of comments from reviews
30/03/2021	1.5	J.G. Moreno (ULR)	Addressed review comments on Section 8
31/03/2021	1.6	L. Leppänen (UH-CS)	Revised Section 9
31/03/2021	2.0	L. Leppänen (UH-CS)	Sent for quality management
01/04/2021	2.1	L. Leppänen (UH-CS)	Minor linguistic and formatting changes
18/04/2021	3.0	A. Doucet (ULR)	Final quality check and minor fixes
12/12/2021	4.0	L. Leppänen (UH)	Updated with NoDaLiDa content
21/12/2021	4.1	G. Hackl (UIBK-DEA)	Internal review
03/01/2022	4.2	M. Weidemann (UROS)	Internal review
12/01/2022	5.0	L. Leppänen (UH)	Final update ready for quality management
15/01/2022	6.0	Antoine Doucet (ULR)	Minor adjustments and submission

Executive summary

This document describes the Reporter component of the NewsEye Personal Research Assistant. The Reporter is a tool for translating the results of the historical newspaper analyses conducted by the Investigator component of the Personal Research Assistant into natural language (e.g. English) descriptions. By doing so, it facilitates a better understanding of the obtained results. The Reporter is constructed as a Natural Language Generation application that takes as input the analyses of other NewsEye tools, augmented with numeric information on their relative importance and surprisingness. This input is fed into a pipeline of Natural Language Generation components that gradually transforms the input into natural language. The individual parts of this pipeline are designed so as to be easily augmented and extended without needing to modify the surrounding parts of the pipeline. Likewise, the pipeline architecture is designed to allow relatively easy augmentation of the process to support new languages by virtue of separating the domain-specific components from the language-specific components where possible.

Contents

Executive Summary 3

1. Introduction 6

2. The NewsEye Personal Research Assistant 7

3. Natural Language Generation 9
3.1. Natural Language Generation as a Process . . . 9
3.2. Methods for Data-to-Text Natural Language Generation . . . 10

4. Requirement Analysis 11

5. Reporter Architecture 12
5.1. Input: Facts and Messages . . . 12
5.1.1. The Fact data structure . . . 14
5.1.2. The Message data structure . . . 16
5.1.3. Fact and Message Generation . . . 16
5.2. Document Structuring . . . 19
5.3. Templates and Template Selection . . . 25
5.4. Lexicalization . . . 27
5.5. Aggregation . . . 29
5.6. Referring Expression Generation . . . 31
5.7. Morphological Realization . . . 33
5.8. Realization into HTML . . . 33
5.9. Link Generation . . . 34

6. Header Generation and Multi-part reports 35

7. Integration with the Personal Research Assistant 36
7.1. Integration with Assistant Control . . . 36
7.2. Integration with Individual Analytical Tools . . . 36

8. Summarization 37
8.1. Text summarization systems . . . 37
8.2. Datasets . . . 38
8.3. Automatic evaluation . . . 39
8.4. Experimental setup . . . 39
8.5. Experimental evaluation . . . 39
8.6. Integration to Reporter . . . 40

9. Evaluation 40
9.1. Technical evaluation . . . 41
9.2. User-centric evaluation . . . 42

10. Conclusions 43

A. Reporter API Description 48
A.1. Endpoints . . . 48

A.2. GET /api/languages . . . 48
A.2.1. Parameters . . . 48
A.2.2. Example Response . . . 48
A.3. GET /api/formats . . . 48
A.3.1. Parameters . . . 48
A.3.2. Example Response . . . 49
A.4. POST /api/report . . . 49
A.4.1. Parameters . . . 49
A.4.2. Example Response . . . 50
A.5. POST /api/report/json . . . 50

B. A Baseline Document Planning Method for Automated Journalism 51

1. Introduction

Historical newspapers collect information about cultural, political and social events in a more detailed way than any other public record. At the same time, analysing the wealth of information in the newspaper archives has traditionally been difficult and time-consuming. The NewsEye project develops methods and tools for effective exploration and exploitation of historical newspaper archives.

The core concept of NewsEye is a set of tools and methods, from text recognition to automated exploration of texts, that improve the users' capability to access, analyze and use the content of historical newspapers, stored in digital libraries (Figure 1).

This document describes the Reporter component of the Personal Research Assistant developed as part of the NewsEye project. The Assistant carries out automated, iterative analysis of corpus content and reports on the results, functioning as the user's intelligent and transparent aid.

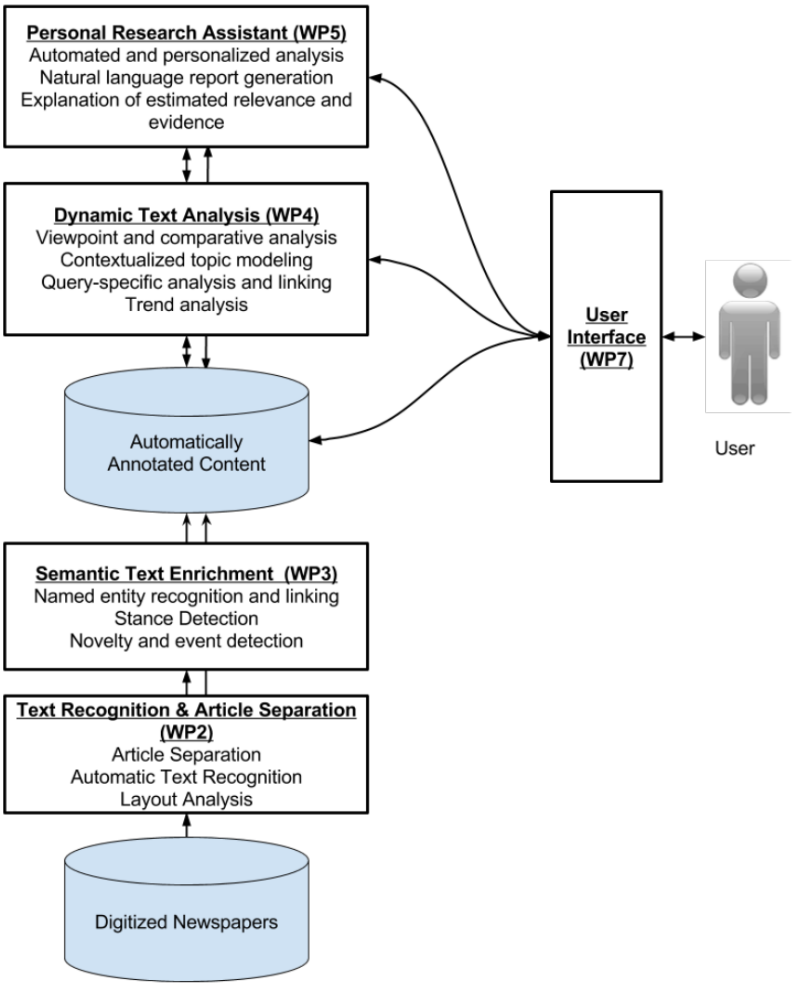

Figure 1: An overview of the NewsEye concept. This document describes the Reporter component of the Personal Research Assistant (Work Package 5).

This document is a complete description of the Reporter at the end of the NewsEye project, meaning that it also contains information about the aspects of the system that remain unchanged from its previous descriptions in the non-public Deliverables D5.2 and D5.4. The version described in this Deliverable has been modified to address user feedback regarding the style of the output texts, supports two new languages (French and German) and the full suite of analytical tools used by the NewsEye Investigator (see Deliverable D5.6). We have improved the structuring of documents discussing multiple datasets and the aggregation and entity name resolution processes. Further changes have been made to e.g. message generation and lexicalization. Finally, the reports can now include optional information that allows a user interface, such as the NewsEye Demonstrator described in Deliverable D7.8, to link between reports and analysis results. An evaluation of the document planning process has been published [1] in the Proceedings of the 23rd Nordic Conference on Computational Linguistics, and is included as Appendix B.

2. The NewsEye Personal Research Assistant

This document describes the Reporter, a part of the larger system known as the NewsEye Personal Research Assistant ('Assistant'). In addition to the Reporter, the Assistant consists of an Investigator component, an Explainer component and an additional Controller component. We next give an overview of these components. A more detailed description of how the Reporter technically integrates with other components in the NewsEye ecosystem is provided in Section 7.

As noted above, the Personal Research Assistant consists of three primary components (Investigator, Reporter and Explainer) and a Controller component. The Controller component has two primary functions. First, it provides an Application Programming Interface (API) for users, especially the NewsEye User Interface (UI, See Work Package 7). This allows outside users to view the Assistant as a single, unified, system so that they do not need to concern themselves with the internal division of labor within the Assistant. The API is used via HTTPS queries and is described in more detail in Deliverable D5.6, which details the present version of the Investigator. Second, as the name implies, the Controller provides a central control mechanism that passes messages and results between the three major subsystems of the Assistant, i.e. the Investigator, the Reporter and the Explainer.

This design facilitates the distribution of the Assistant components over multiple virtual or physical machines if such a distribution would become needed due to increasing amounts of users. Modifying the Assistant so that a single Controller instance acts as a load balancer and distributor of work to multiple instances of the subcomponents, while still requiring some programming and engineering effort, would be relatively simple following standard approaches used in many other online services. In other words, as the Reporter itself is 'pure' (i.e. running the Reporter on some input has no side effects that would have an effect on a future run, and the produced text is always the same for a specific input) a relatively simple extension of the Controller would be to have it spin up virtual machines running instances of the Reporter in response to incoming API requests, using e.g. Docker.

The Investigator component, in broad terms, autonomously performs a series of queries over a newspaper corpus using different tools provided by Work Packages 3 and 4 to identify potentially interesting factors from the data. The Investigator is detailed in Deliverable D5.6. The analytical results obtained by the Investigator are passed via the Controller to the Reporter component described in this Deliverable. The controller also has the ability to cache and reuse investigation results using a database.

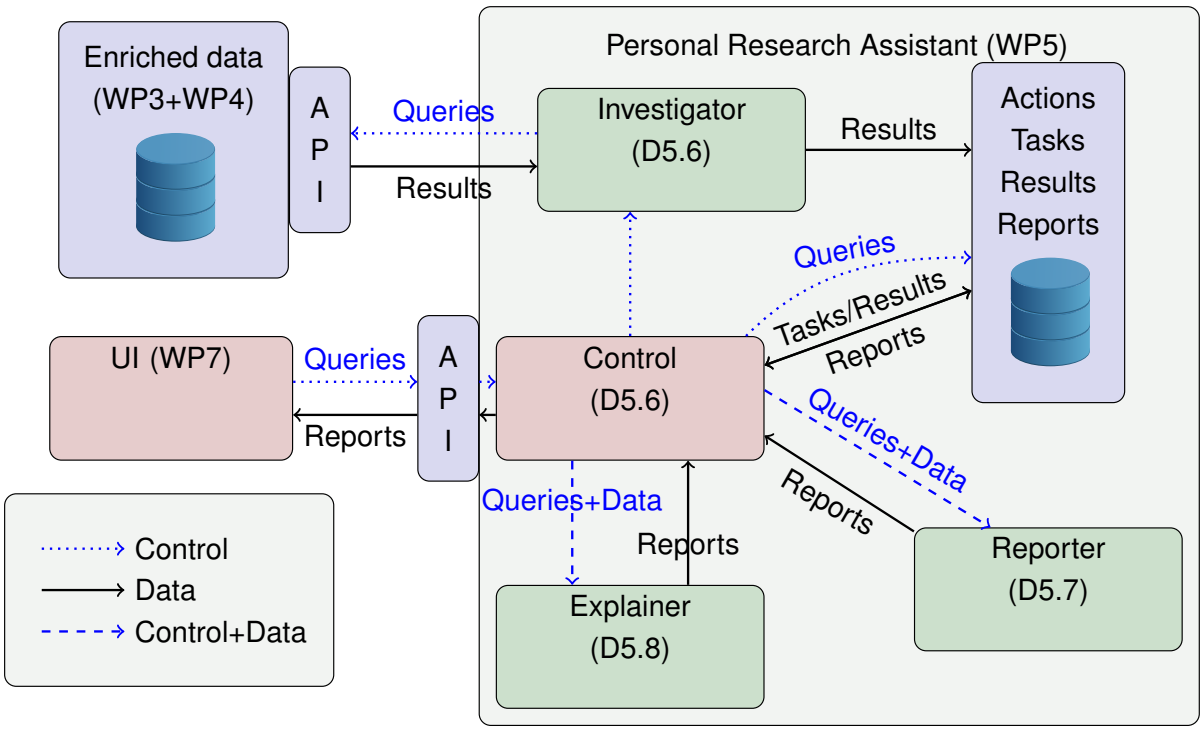

Figure 2: Flow of requests and data between the components of the Personal Research Assistant and the associated components developed in Work Packages 3, 4 and 7.

Having received the analytical results from the Assistant Controller, the Reporter then transforms the results into natural language expressions. This transformation process is discussed in significant detail in the rest of this Deliverable. The resulting natural language document is then returned to the Controller. The Controller then sends the document to the party that requested it. In the most likely scenario, this is the NewsEye User Interface that displays it to the end user.

Finally, the Assistant also contains an Explainer component. Whereas the Reporter describes *what* the Investigator found in the corpus, the Explainer described *how* those findings were obtained and *why* the Investigator believes them to be of interest. In other words, whereas the Reporter describes the end result of a process, the Explainer describes the process itself. The final version of the Explainer component is described in Deliverable D5.8.

In addition to this, as noted above, the Assistant also contains a database component which caches analysis results obtained from the Investigator, reports obtained from the Reporter, the explanations generated by the Explainer and other necessary data. An overview of the Personal Research Assistant's architecture is presented in Figure 2.

In terms of the division of labor between the Investigator and the Reporter, it is notable that the Reporter does not conduct any additional investigative work or in any way enhance the results obtained from the Investigator. It merely produces a natural language expression detailing the findings of the Investigator. The only prerogative the Reporter has over the analysis results is its ability to limit the report to a certain length. That is, if the analysis results in an amount of 'potentially interesting' results that is too large to reasonably produce a concise report out of, the Reporter can and will omit some details from the final report.

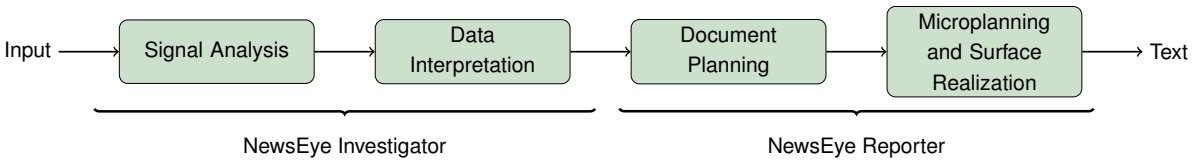

Figure 3: Alignment of tasks from Reiter's reference architecture [13] and the Assistant components.

3. Natural Language Generation

The general task conducted by the Reporter is known as Natural Language Generation, or NLG for short. More specifically, the Reporter is performing 'data-to-text NLG', where 'data' refers to structured data. That is, the Reporter is not designed to ingest unstructured data, such as raw text. In this section we first give an overview of (data-to-text) NLG in general as well as how it can be viewed as a series of subtasks, followed by a description of how the subtasks common to data-to-text NLG can be completed.

3.1. Natural Language Generation as a Process

A large number of data-to-text systems have been reported in the literature, in different domains, with varying types of input data [2]. For example, BabyTalk [3] is an NLG system that generates medical reports from sensors monitoring babies in Neonatal Intensive Care Units. Hallett and Scott [4] describe a system for generating reports from events in medical records. Several systems have been developed that generate weather forecasts from the output of weather computer simulation models [5, 6, 7]. Others still generate summaries from employment statistics [8, 9]. In addition to these efforts, several commercial NLG systems exist in a number of domains. Among the larger commercial players that provide NLG products and/or services are Automated Insights, Arria, AX Semantics, Narrativa, Narrative Science, and Yseop [10]. As demonstrated by the amount of commercial entities working in the domain, NLG technology is of increasing interest even without the academia, for example in the newsroom, even if it is not always clear to the non-technical stakeholders how to best employ the technology [11].

General data-to-text NLG systems normally involve three processes: deciding what to say (content determination), how to organize it (document and sentence planning), and how to express it (surface realization) [12]. A reference architecture for data-to-text systems is presented by Reiter [13]. In this reference architecture, Reiter builds on the general NLG architecture by dividing the data-to-text generation process into the following subcomponents: signal analysis, data interpretation, document planning, and microplanning and realization. Figure 3 demonstrates how these components align with the different sub-components of the Personal Research Assistant.

The signal analysis stage analyses numerical and other input data, looking for patterns and trends, and the data interpretation stage identifies more complex and domain-specific messages. Here, messages are pieces of information that are meaningful in isolation and could be conveyed to the reader via the final text document produced by the data-to-text NLG system. In terms of the Personal Research Assistant, these processes are viewed as being completed by the Investigator (see Deliverable D5.6, released concurrently with this Deliverable) and as such are not explored further in this document.

Document planning is a stage that ingests the so-called 'messages' produced by the previous stage and organizes them into a structure that defines in which order they should appear in the final document. This initial version of the *document plan* is based solely on the information of the messages and can be

modified later in the process for better linguistic fluency. For example, if the document planning phase decides to place two pieces of information one after another in the document plan, a further stage of the generation process might place a third piece of information between them if it makes the resulting text more fluent.

The following stage, microplanning and realization, takes as input the document plan and produces the final text output. This stage is often further divided into several subtasks that are treated separately. These subtasks include selecting the basic phrases that are used to express the individual pieces of information, aggregation, lexicalization and surface realization.

Finally, we emphasize that the above characterization is more of an aide for reasoning about the types of decisions NLG systems have to complete. Previous NLG systems have employed a wide variety of techniques that can make the distinctions between the aforementioned phases of the generation process fuzzy or even remove them altogether [14].

3.2. Methods for Data-to-Text Natural Language Generation

In addition to different ways of dividing the larger generation tasks to smaller subtasks, there are also several competing methods for completing said subtasks [14]. In broad terms, we identify two main approaches: rule-based approaches and trainable approaches.

Rule-based systems are based on handcrafted rules, corpus analysis and expert consultations [15]. They are usually more robust than trainable approaches and are widely used in industry and for commercial purposes. As the rule-based approach is finely controlled, the output can be guaranteed to be more understandable by humans. Rules also provide relative high guarantees of correctness, and in case of errors, can be easily corrected by modifying the system's source code. At the same time, rules are limited, especially when the domain complexity increases and the generation of the rules can be an expensive effort requiring significant input from domain experts.

Trainable approaches, such as neural networks, reinforcement learning, or Hidden Markov Models are more flexible, easier to develop, and more domain-independent. However, one of the challenges of these trainable approaches is the lack of sufficient quantity of aligned datasets that can be used to derive rules or train the NLG system [15]. Even when the data is available, the expected output text is often not aligned with the input data, and thus cannot be used directly for the development of an NLG system [16].

At the onset of the NewsEye research process, we identified that the state of the art in these trainable approaches also seemed to suffer from a series of further problems that were relevant for the NewsEye context. First, purely trainable approaches struggled to reach the linguistic depth of their competitors [17], with even the then-most-recent trainable end-to-end architectures failing to conclusively outperform rule-based approaches even in a relatively simple and constrained generation task when evaluated by humans [18]. Second, most trainable approaches suffered from a lack of transparent generation process. This chiefly manifested in the difficulty of detecting and correcting system errors: it was exceedingly difficult to guarantee any specific level of correctness for an NLG system based on neural networks, and in case of problems in the system it was not possible to conduct surgical modifications. Rather, especially in case of (deep) neural networks, the only solution was to further train the system with more training data. Whether this retraining and fine-tuning results in some unknown pathological behaviour in some corner cases would be difficult if not impossible to determine. Third,

empirical evidence indicated that such systems were – and continue to be – prone to overfitting to the training data, which manifests as '*hallucination*', where the system produces output that is not grounded in the underlying data [19]. These problems were made more complicated by the fact that automated evaluation of an NLG system's quality was – and continues to be – an unsolved problem, with evidence suggesting that the most popular automated metrics fail to properly correlate with the judgments of human evaluators [14, 18, 20, 21].

Our interpretation of the current state of the art in NLG at the onset of the NewsEye research project was thus that trainable approaches were mainly ready for real-world use in situations where either the produced texts were very short (i.e. scenarios similar to the E2E Challenge described by Dušek, Novikova, and Rieser [18]) or in scenarios where even major mistakes in individual pieces of output are not problematic. While many of the problems identified above have seen significant scholarly attention, with attention being directed especially towards the hallucination problem [22, 23], the present state of the art in NLG has not significantly modified the above analysis. For example, hallucination continues to plague even state-of-the-art neural systems [24, 25]. As such, we do not believe the present state of the art in NLG is sufficiently different from that at the onset of the NewsEye project, and thus our requirement analysis (see below) conducted at the onset of the project remains valid.

4. Requirement Analysis

The NewsEye Personal Research Assistant is intended to be used for exploration of historical newspaper corpora. Part of the intended user base is formed by academics such as historians and social scientists. While the academic users are likely to independently verify any findings reported by the Reporter, any mistakes are going to erode trust in the system and make the users less likely to use it in the future. Furthermore, other user demographics such as lay-historians are not necessarily so thorough in their investigations and might skip the manual verification of the presented results. As such, the Reporter component has a significant *correctness* requirement in that it does not misrepresent the analyses conducted by the Investigator. This requirement speaks against using a trainable approach to NLG.

The NewsEye Personal Research Assistant is also intended to be easily *extensible* so that it can take advantage of new, improved, analytical tools as they become available. Adding such a new tool should be straightforward and should not invalidate previous work. This requirement speaks towards a modular system, where each analytical tool is accompanied by a small module that can be incorporated into the Reporter and is in charge of any decisions specific to that one analytical tool's output.

The Assistant has to be *explainable*. That is, it cannot function as a black box where the internal workings are either completely opaque or so complex as to not be understandable by a human. This requirement speaks against using a trainable approach based on, e.g. neural networks.

The Reporter must be able to produce reports in *multiple languages*. While this requirement does not specifically *require* any specific approach, taken together with the *extensibility* requirement it speaks for a modular approach. A modular system can be constructed so that the language-independent parts of the generation process can be shared by the different languages. This further improves the extensibility of the system by allowing the system to be extended to new languages without having to duplicate all of its components.

As a summary, we identify the following requirements and their implications for the system:

- Correctness - Suggests a rule-based approach
- Extensibility - Suggests a modular approach
- Explainability - Suggests a rule-based approach
- Multilinguality - Suggests a modular approach

As a whole, the requirement analysis suggests that the Reporter should be a modular system based on human-produced rules. This analysis is further supported by the lack of any suitable training data set that would be required for a trainable approach to be feasible.

At the same time, the requirement analysis does not completely forbid the use of trainable methods as *parts* of the larger system in settings where they are unable to significantly affect the correctness of the output. Early results in other domains indicate that dividing the unified, end-to-end, neural NLG model into separate, but still neural, subcomponents increases the performance of the system [24, 26]. While the limitations of such models are far from known, even these early successes indicate that a hybrid system employing both rule-based and neural modules could be successful. This further suggests a modular architecture that facilitates the inclusion of neural components if they are developed.

5. Reporter Architecture

Based on the requirement analysis described above, it was determined that a principally rule-based, modular, Natural Language Generation (NLG) system was the best fit for the Reporter. As such, we decided to base our system on a modification of the multilingual news report generator previously produced by the University of Helsinki Department of Computer Science [27]. The modified architecture (see Figure 4) is formulated as a pipeline where raw data and relevant parameters are fed in at the start. This input is then processed through a pipeline of individual components, where the components of the pipeline each modify the input towards the final output. At the end of the pipeline, a finished natural language text document is produced as the output. In the next sections, we will step through the generation process, discussing each pipeline component in turn.

5.1. Input: Facts and Messages

The primary input to the system are the results of the analyses conducted by the Investigator component. These results are provided to the Reporter in the JavaScript Object Notation (JSON) format via an HTTP(S) request. The JSON format [28, 29] is an industry standard format for moving data between services using HTTP(S). Figure 5 shows an excerpt of what the system input looks like.

The Reporter refers to information in two primary formats: as *facts* (see Section 5.1.1) and as *messages* (see Section 5.1.2). A fact represents both an immutable point of data (e.g. a numeric value obtained as an analytical result) and its associated metadata, i.e. what information it describes. It is a minimal piece of information that is expressible to a human reader and the smallest informational unit used by the system. The message wraps a fact and provides a place for any mutable data that needs to be associated with a specific fact for the generation process to succeed.

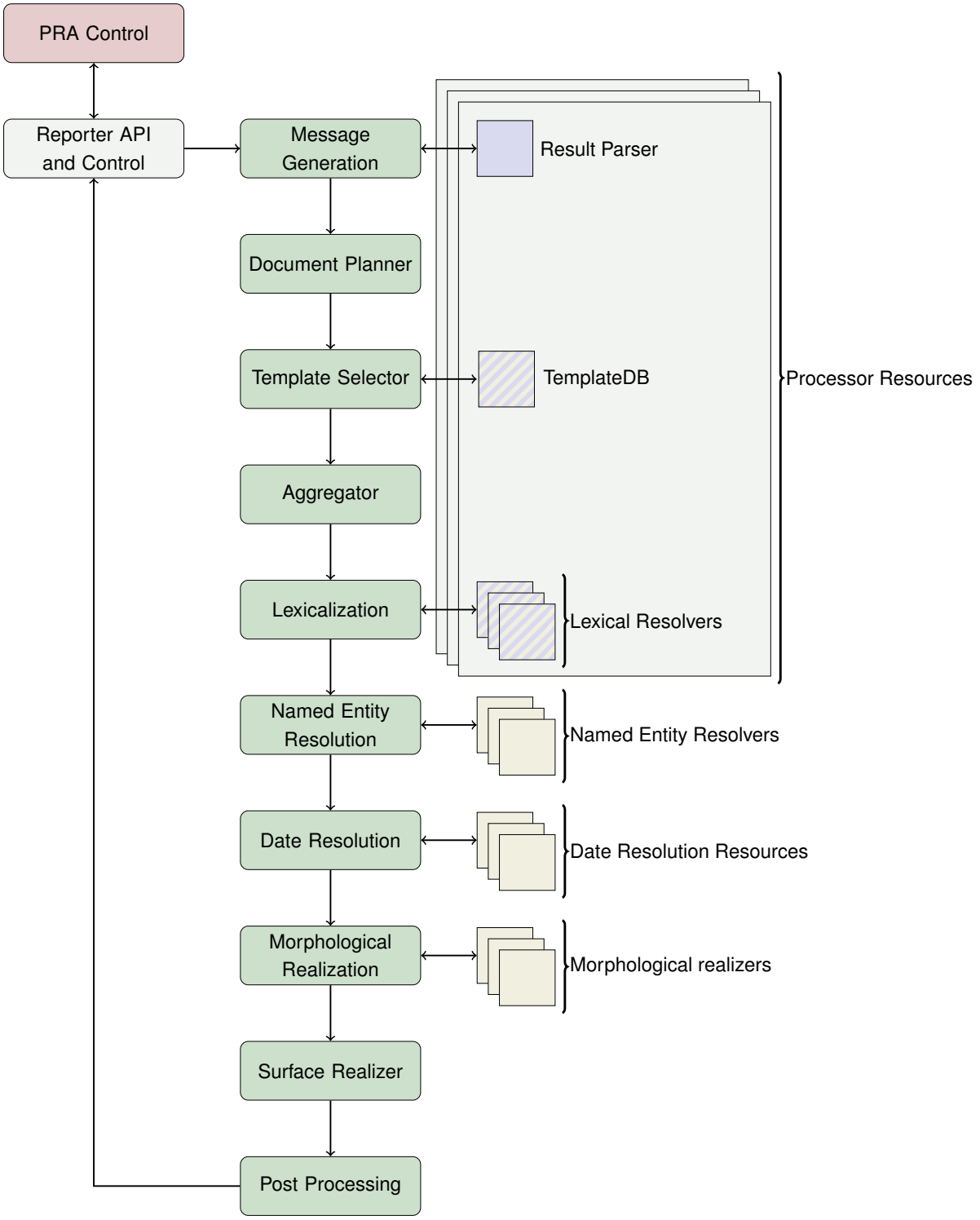

Figure 4: High-level architecture of the Reporter. Sharp-cornered elements represent various resources available to the Reporter. Sand-colored elements (e.g. Morphological realizers) are language-specific but not directly related to any specific analytical tool. Sapphire-colored elements (e.g. Result Parser) are specific to an analytical tool, but not to any language. Dashed elements (e.g. TemplateDB) are specific to both. The green middle column forms the main NLG pipeline.

5.1.1. The Fact data structure

The fields of the Fact data structure are detailed in Table 1, together with example values. The fields `corpus` and `corpus_type` define what corpus the fact is discussing. It is important to note here, that the concept of 'corpus' is in this case relatively flexible and refers, in practice, to any addressable and investigable collection of documents. Most trivially, this is for example the complete ONB newspaper corpus. More likely, however, it is a collection of documents defined by some user query, such as 'all articles that contain the keyword "femme".' These fields would also contain any temporal information specific to the *corpus*, i.e. that the corpus being analyzed contains articles from a specific decade.

The fields `timestamp_from`, `timestamp_to` and `timestamp_type` enable the inclusion of additional temporal information specific to the *analysis*. This is primarily relevant to, for example, time series analysis where the system could want to report that in a corpus spanning years 1900 to 1914, the occurences of a search term peak on years 1908 and 1909. Here, the corpus span would be encoded into the `corpus` field whereas the start and end years of the peak would be encoded in the `timestamp` fields.

```
[
  {
    "processor": "ExtractFacets",
    "search_query": {
      "q": "femme"
    },
    ...,
    "task_result": {
      "result": {
        "PUB_YEAR": {
          "1935": 6420,
          "1936": 4502,
          ...
        },
        "LANGUAGE": {
          "fr": 114385,
          "fi": 134
        },
        "NEWSPAPER_NAME": {
          "l_ouvre": 114385,
          "uusi_suometar": 134
        }
      }
    }
  },
  ...
]
```

Figure 5: An example of the analysis results provided by the Investigator. Note that the data structure has been edited for length and that both metadata and parts of the results have been omitted.

Field	Example value
corpus	[q:femme]
corpus_type	query
timestamp_from	None
timestamp_to	None
timestamp_type	all_time
analysis_type	ExtractFacets:absolute_count
result_key	[PUB_YEAR:1936]
result_value	4502
interestingness	1

Table 1: The fields of the 'fact' data structure and an example of their possible values. This fictional example fact would correspond to the idea that a corpus consisting of all the news articles that match the query 'femme' contains 4502 articles published during the year 1936.

The `timestamp_type` describes how to interpret the timestamps in the `from` and `to` fields. For example, the value pair `timestamp_from = 2019/01/01 00:00` and `timestamp_to = 2019/12/31 23:59` could be reasonably reported as '*during the year 2019*', as '*between January and December 2019*' or as '*Between 01/01/2019 and 31/12/2019*' depending on what the context is. The use of a single `timestamp_type` field, rather than a separate field for both values, enforces that the timestamps are expressed in the same format. This simplifies the generation process as we explicitly disallow many complex and hard-to-interpret expressions such as 'between the 1860s and December 1st of 1973'.

The fields `analysis_type`, `result_key` and `result_value` together describe a singular value obtained by the Investigator during conduction of some analysis. For example the values `analysis_type = ExtractFacets:absolute_count`, `result_key = [PUB_YEAR:1936]` and `result_value = 4502` together describe the idea that a corpus consisting of all the news articles that match the query 'femme' contains 4502 articles published during the year 1936. Modifying the field `analysis_type` to instead contain the value '`ExtractFacets:percentage`', together with modifying the field `result_value` to contain the value '13', would then indicate that 13 % of the articles in the corpus were published during 1936.

Finally, the fact contains an `interestingness` field that describes how interesting the fact is in the view of the Investigator. This information is crucial in the following stages where the system uses it to determine what facts should be part of the document and in what order. The Reporter only mandates that the `interestingness` value is a positive number so that a larger number indicates a more interesting fact and a zero indicates something that is 'absolute not interesting'. The determination of this value, as well as whether there is some 'maximal' value, is left to the Investigator.

We note that, as demonstrated in Table 1, some of the values are presented as *tags*. The system places no constrictions on how these tags are formulated, thus enabling additional meta-information to be added to the results on a per-analysis basis. Similarly, the `analysis_type` field encodes explicitly the analytical tool behind the fact as its first element, with the rest of the field's value consisting of a context-specific hierarchy describing the relevant parameters and metadata, each separated by a colon, ':'. This formulation is used later on, in the content determination phase, to assess the degree of similarity between multiple analytical results.

Field	Description
facts	The facts related by this message. By default contains only a single fact, but can contain multiple, e.g. as a result of aggregation.
template	The template associated with the message. Initially empty, gets assigned during template selection.
score	A potentially modified interestingess value.

Table 2: The fields of the 'message' data structure and a description of their contents.

During the processing performed by the Reporter, the fact itself is an immutable result of the analysis. In other words, the Reporter never modifies the fact itself once it has been created. This ensures that the underlying analysis results remain unchanged through the generation process.

5.1.2. The Message data structure

As noted above, the fact data structures are considered immutable and the generation is not to modify them in any way. In fact, they are technically implemented in a way that makes it impossible for them to be modified by accident. This ensures the underlying information within the generation always stays true. At the same time, the system needs to attach information to the facts, such as what *template* (see Section 5.3) is to be used to express the fact. For this reason, we encapsulate the immutable facts in mutable data structures called 'messages'. In the present version of the Reporter, the message data structures contain the fields shown in Table 2.

Beyond the mutability, another notable distinguishing feature between fact and messages is that a single message may contain multiple facts. This can happen as a result of, e.g. the aggregation phase where multiple templates are combined into a single template which then expresses multiple facts.

The field *score* allows for modification of the underlying fact's interestingness value without overwriting the original value. This is useful, for example, in the document structuring process where we wish to consider not only how interesting a fact is, but also how well it fits a specific context. All the score fields are, however, initialized to the underlying facts' interestingness values.

The message can also be extended to transfer other information relevant to the generation process. For example, in a previous system we have used the message to transfer information regarding the 'polarity' of the fact, i.e. whether it is a positive or a negative thing, to determine which conjunctions are to be used between two facts.

5.1.3. Fact and Message Generation

The facts and messages are generated from the analytical results provided by the Investigator by the message generator. It ingests the JSON formatted analysis report and outputs the messages and facts, which are in turn ingested by the document planner.

Notably, the specifics of how an analysis result is to be parsed is dependent on the precise analytical tool used by the Investigator (see Deliverable 5.6). This means, in practice, that the parsing functionality of the message generator needs to be adjusted every time the tools used by the Investigator are altered, or when a new tool is integrated into the Investigator.

As such, the message generator is constructed so that it delegates the parsing to a set of individual parsers, each of which contain a method for identifying the relevant subset of results it can parse and returning the resulting Facts to the main program. These individual parsers are depicted in Figure 4 as the small boxes with the label 'Result Parsers'.

The processing is done on the level of the output of the analyses done by the Investigator. For example, the Investigator might implement an analysis called '*ExtractFacets*' the result of which would be a data structure such as that presented in Figure 5, above. Note how the input of the message generation component is a list of one or more such results. The message generator component then applies each result parser known to it to each of the results, collecting the generated messages. This procedure is described below in pseudocode as function GENERATEMESSAGES in Algorithm 1.

Algorithm 1 Pseudocode describing the relation between the analytical results (see Figure 5) and the result parsers. Note how each analytical result must be parsed in isolation, but can be parsed by multiple ResultParsers.

function GENERATEMESSAGES($AnalyticalResults$, $ResultParsers$)
  $Messages \leftarrow []$
  for all $AnalyticalResult \in AnalyticalResults$ **do**
    for all $ResultParser \in ResultParsers$ **do**
      $NewMessages \leftarrow ResultParser.apply(AnalyticalResult)$
      $Messages.extend(NewMessages)$
    end for
  end for
  return $Messages$
end function

The upside of this formulation of the message generator is that it allows for significant decoupling of the Reporter's core functionality from the specifics of individual tools, thus enabling easier modifiability and simpler integration of any future tools. This process only requires a minor amount of integration work from a programmer.

By default, there is a one-to-one correspondence between a type of analysis conducted by the Investigator and the associated Result Parser. However, as can be observed from Algorithm 1, this is not a strict requirement and one-to-many relations can be implemented where necessary. At the same time, the formulation of Algorithm 1 prevents the inclusion of many-to-one result parsers, i.e. parsers that observe the results of multiple analytical tools to generate their messages. This discourages large-scale post-hoc analysis conducted within the Reporter.

One-to-Many relations are potentially beneficial especially in situations where some relevant information is not directly available from the result, but rather must be separately calculated. As a more concrete example, the data structure described in Figure 5 lacks a field denoting the total number of articles. This number can easily be retrieved by summing over any one of the '*PUB_YEAR*', '*LANGUAGE*' or '*NEWSPAPER_NAME*' listings using a separate Result Parser. This, however, begins fairly quickly to

encroach of the responsibilities of the Investigator and such additional analysis should be implemented primarily in the Investigator, as noted above, and is discouraged.

At the present, the Reporter conducts a very limited amount of such post-hoc analysis. For example, in the case of `TrackNameSentiment` processor's result, which describes how the sentiment towards an entity has changed over some time frame, the relevant message generator extracts the minimal, maximal and mean sentiments, as well as the number of years within the time frame wherein the entity is discussed in the corpus. Without this post-hoc analysis, the Reporter would be reduced to reporting only a collection of individual yearly datapoints.

As noted above, such post-hoc analysis is not philosophically in the purview of the Reporter and should occur at the Investigator component. As such, we have refrained from implementing such post-hoc analyses in the other message generators used by the Reporter. The incorporation of post-hoc analysis in the `TrackNameSentiment` tool simply demonstrates how such capabilities *could* be added, if future developments would warrant a change in the responsibilities of the Reporter and the Investigator.

While the message generation process is largely language agnostic (i.e. independent of what language is being produced), a small exception is made to account for some entities in analyses such as `TrackNameSentiment` (which details how the sentiment towards and entity has changed over time) and `ExtractNames` (which identifies named entities from the corpus). Both of these output identifiers such as `entity_LOC_Q1757` that identify a named entity in a language-agnostic manner. To enable translation of these abstract identifiers to language specific phrases (such as 'Helsinki' in case of `entity_LOC_Q1757`), the Reporter needs a dictionary of the relevant entities' names in the languages supported by the Reporter.

This information could be obtained by querying the SOLR database (See 'Enriched data (WP3+WP4)' in Figure 2), but this would necessitate making the Reporter aware of the relevant APIs and network addresses, thus massively increased the network topology's complexity. To avoid this increased complexity, the Investigator instead queries the SOLR database for the relevant dictionary and provides it as part of the Explainer's input. To simplify the generation process, instead of passing the dictionary through the complete NLG stack to the Named Entity Resolution module (See Figure 4), the message generation module replaces the language agnostic identifier with a language specific expression as part of the message generation process.

This replacement operation raises a non-trivial question in cases wherein the Reporter is discussing documents in one language (e.g. Finnish) in a report written in another language (e.g. English). Namely, should entities be referred to in the language used in the text, or in the language of the report? In other words, should the document refer to the city of Vyborg as 'Vyborg' (using the language of the report); as 'Viipuri', as it likely appears in the Finnish language text; or using all as 'Viipuri (Vyborg)'. While all three options have upsides and downsides, we have elected to use the language of the report as the overriding option. This was done to account for the observation that while a report is always monolingual (i.e. has a unique correct entity name), a corpus being analyzed can consist of documents in multiple languages. In the case of a multilingual corpus, the other methods could produce extremely unwieldy natural language expressions as worst-case behaviour.

The output of the message generation process is an unordered set of message objects, each containing a single immutable Fact.

5.2. Document Structuring

The message data structures, which in turn contain the fact data structures, are provided as input to the next component in the pipeline, the document planner. The facts and messages, obtained originally from the Investigator, describe what *could* be described in the report before any practical limitations, such as the length of the resulting document, are taken into account. The document planning stage determines which facts to actually express and how to structure and order them in a report given these additional constraints.

The output of this phase of the process is a tree structure that largely corresponds to the paragraphs (or other equivalent high level structures) of the result document. This structuring is driven solely by the contents of the messages, with the bulk of the effort being based on the *interestingness* scores returned by the Investigator.

During this data-driven document planning, the system assigns facts in the document plan one of two roles: '*nucleus*' or '*satellite*'. The terminology 'nucleus' and 'satellite' is most commonly associated with the Rhetorical Structure Theory (RST) [30], where a nucleus contains the most important, information and are to a degree independent, while satellites provide additional information about the nuclei and are usually not meaningful without the relevant nucleus. Our system combines facets of the RST with the orbital theory of news structuring as presented by White [31], who uses the same nucleus-satellite terminology to describe higher-level structures. White describes hard news as consisting of an orbital structure where a lead nucleus, described usually in the headline and the first paragraph, is supported by orbiting satellites adding further information to the story. White notes that these satellites can often be ordered relatively freely without significantly affecting the story. We modify White's theory by considering each paragraph as consisting of a separate orbital structure of a nucleus and satellites phrases, thus bridging between White's high-level nucleus-satellite structures and the low-level RST nucleus-satellite-relations by considering the overall document to be a recursive structure consisting of nucleus-satellite relation.

The procedure that creates the document plan is described as pseudocode in Algorithm 2. The generation progresses paragraph-by-paragraph, first selecting a suitable nucleus based on the previously selected nuclei, and then selecting suitable satellites for that nucleus to fill in the paragraph.

The system bases its decisions on two factors: the interestingness scores of the messages and a concept of thematic similarity, which describes how similar the topics of two arbitrary messages are. The goal is to produce a text that contains as interesting messages as possible, while also enforcing a level of coherence into the document.

In the case of the first paragraph, the `SelectNucleus` procedure simply selects the most interesting fact in the system input (i.e. the Investigator's output) as the *nucleus* of the first paragraph. This is a special case, and later nuclei are selected using a more complex process, described later-on.

To this nucleus, additional supporting facts or *satellites*, are added, as determined by the SELECTSATELLITES procedure, described in pseudocode in Algorithm 3. These satellites are selected from among all available, so far unused, messages one-by-one. After each selection, the available, so far unused, satellites' interestingness scores are recalculated to reflect the satellites' similarity to both the previously selected satellite and the nucleus of the paragraph. That is, when selecting the first satellite, the similarity is measured against the nucleus only, whereas for the third satellite, the similarity

Algorithm 2 Pseudocode describing generation of the document plan from messages.

function GENERATEDOCUMENTPLAN($Messages$)
  $Root \leftarrow newDocumentPlanNode$
  $SelectedNuclei \leftarrow []$
  while $True$ **do**
    if reached maximum length **or** $Messages = \emptyset$ **then**
      return $Root$
    end if
    $Node \leftarrow newDocumentPlanNode$
    $Nucleus \leftarrow$ SELECTNUCLEUS($Messages$, $SelectedNuclei$)
    if $Nucleus =$ null **or** $Nucleus$ is not sufficiently interesting for inclusion **then**
      return $Root$
    end if
    $Satellites \leftarrow$ SELECTSATELLITES($Nucleus$, $Messages$)
    $Node.children = [Nucleus]$
    $Node.children.extend(Satellites)$
    $SelectedNuclei.insert(Nucleus)$
    $Root.children.insert(Node)$
  end while
end function

is measured against the second satellite and the nucleus. The similarity between two messages is a determined using a combination of ANALYSISSIMILARITY (for the nucleus and the previous satellite) and CONTEXTSIMILARITY (for the previous satellite only) procedures. Both procedures produce a weight that is then used to update a candidate message's score.

The intuition behind this approach is to maximize the interestingness of the paragraph's contents, while also enforcing a certain level of coherence in the text. By continuously measuring against the previously selected satellite, the procedure allows for some *thematic drift* within the paragraph. That is, the theme of the paragraph can evolve over time. At the same time, the inclusion of the similarity measure against the nucleus also ensures that the theme does not drift *excessively* far from the original theme of the paragraph.

The CONTEXTSIMILARITY procedure considers two messages to be more similar in *context* if they share the values of their underlying facts' `corpus`, `timestamp_to`, `timestamp_from`, and `result_key` fields. For every field for which the two messages' field values are the same, a similarity value (initially 1) is multiplied by a weight. These weights are set per-field, which in turn enables the system to consider certain types of similarities to be more important than others for the purposes of document structuring. At the present, the weight for the `corpus` field is 1.5; the weights for the `timestamp_from` and `timestamp_to` fields are 1.1; and the weight for the `result_key` field is 5. As such, the `result_key` field dominates the selection process, while still accounting for the similarities of the other contextual fields. We note that these weights are tuneable hyperparameters, and can be further modified to account for user feedback.

The ANALYSISSIMILARITY procedures two messages to be similar in *analysis* based on the `analysis_type` field. As noted in Section 5.1.1, the field contains a colon-separated hierarchy of labels describing the analysis behind the fact. The first item is always the name of the analytical tool, while subsequent items contain additional information. For example, the fields of three distinct facts could contain the value

Algorithm 3 Pseudocode describing how satellites are selected for a paragraph

```
function SELECTSATELLITES(Nucleus, Messages)
    SelectedSatellites ← []
    prev ← Nucleus
    while True do
        if maximum satellite count reached then
            return SelectedSatellites
        end if
        for all m ∈ Messages do
            m.score ← m.score× ANALYSISSIMILARITY(m, prev)
            m.score ← m.score× ANALYSISSIMILARITY(m, Nucleus)
            m.score ← m.score× CONTEXTSIMILARITY(m, prev)
        end for
        FilteredMessages ← FILTERBYINTERESTINGNESS(Messages)
        if FilteredMessages = ∅ then
            if minimum satellite count reached then
                return SelectedSatellites
            else if Messages ≠ ∅ then
                FilteredMessages ← Messages
            else
                return SelectedSatellites
            end if
        end if
        NewSatellite ← arg max_{m∈FilteredMessages} m.score
        SelectedSatellites.append(NewSatellite)
        Messages.remove(NewSatellite)
        prev ← NewSatellite
    end while
end function
```

`ExtractFacets:absolute_count` for fact F_1, the value `ExtractFacets:percentage` for fact F_2 and the value `GenerateTimeSeries:absolute_count:max_val` for fact F_3. Intuitively, F_1 and F_2 are thematically closer than F_1 and F_3. We model this observation into a measure of similarity between two facts A and B as

$$sim(A, B) = \frac{2p(A, B)}{\ell(A) + \ell(B)} \tag{1}$$

where $\ell(A)$ is the length – in colon-separated units – of A's `analysis_type` field. That is, $\ell(F_1) = 2$ and $\ell(F_3) = 3$. Similarly, $p(A, B)$ is the length – in colon-separated units – of the shared prefix between A and B's `analysis_type` fields. For example, $p(F_1, F_2) = 1$ whereas $p(F_1, F_3) = 0$. Applying the formula to various pairs of `analysis_type` fields, we observe the behavior shown in Table 3, which matches our intuition of the degree of similarity.

Presented in terms of a string distance, the above metric is the fraction of the shared prefix out of the total length of the inputs. This formulation also accounts for potentially different length inputs. This is required, for example, in the case of the `TopicModelDocSetComparison` tool, which compares two sets of documents using a topic model. There, the `analysis_type` values range from 3 to 4 colon-separated units.

A	B	$\ell(A)$	$\ell(B)$	$p(A,B)$	sim(A, B)
a:b:c	a:b:c	3	3	3	1
a:b:c	a:b	3	2	2	0.6
a:b:c	a	3	1	1	0.5
g:e:f	a	3	1	0	0
g:e:f	a:b:c	3	3	0	0
a:b:c	a:b:x	3	3	2	0.6
a:b:c	a:x:y	3	3	1	0.3
a:b	a:x	2	2	1	0.5
a:b	a:x:y	2	3	1	0.4
a	x	1	1	0	0

Table 3: Examples of the behavior of the analytical similarity metric.

As noted in Algorithm 3, the scores are recalculated after every new satellite has been selected to account for the effect of that satellites inclusion on the similarities. Satellites are added in this manner until the paragraph is considered full (by virtue of reaching a configurable maximum length) or the system runs out of 'sufficiently interesting' facts that pertain to the theme of the paragraph, as determined by the procedure `FilterByInterestingness`. Here, the messages' scores are compared to both an absolute and a relative threshold value. The message is left in the candidate set if the score is either greater than than the absolute threshold value $t_{abs} = 0.2$, or greater than the score of the paragraphs nucleus multiplied by $t_{rel} = 0.5$. Both t_{abs} and t_{rel} are tuneable hyperparameters and can be trivially modified based on user feedback.

The procedure also accounts for a tuneable minimal satellite count. If there are no more sufficiently interesting messages, but the minimal threshold has not been reached, the threshold of 'sufficiently' is relaxed so that even very uninteresting messages can be used if available, until the minimal paragraph length is reached. Based on experimentation, we set the minimal satellite count at 4, meaning that almost all paragraphs should be at least 5 messages long (4 satellites + 1 nucleus). The minimal length can be ignored only if the messages completely run out during the generation process.

After the satellites have been selected, a new document plan node is constructed out of them and the nucleus, which is then added to the overall document plan.

After this, the nucleus of the next paragraph is selected using the NEXTNUCLEUS procedure. This procedure is described as pseudocode in Algorithm 4.

In terms of building the document, an important goal is to also maximize the overall coverage of the results described to the user across the paragraphs. Whereas, with the satellites, we sought to maximize the semantic similarity between the satellites, the reverse holds for the nuclei. In other words, we want the different paragraphs to discuss as different things as possible.

To this end, Algorithm 4 seeks to enforce the requirement that each paragraph's nucleus must be a message about a so-far undiscussed analytical tool. This, however, causes a problem when **all** the results available for discussion are from a single analytical tool. For this reason, we specifically allow that a previously discussed analysis can be the nucleus of a new paragraph in the case where no other options are available.

Algorithm 4 Pseudocode describing how the next nucleus is selected.

function SELECTNUCLEUS($SelectedNuclei$, $Messages$)
  if $SelectedNuclei = \emptyset$ **then**
    return $\arg\max_{m \in Messages} m.fact.interestingness$
  end if
  $DiscussedAnalyses \leftarrow [\text{NAMEOFANALYTICALTOOL}(n) | n \in SelectedNuclei]$
  $FilteredMessages \leftarrow [m \in Messages | \text{NAMEOFANALYTICALTOOL}(m) \notin DiscussedAnalyses]$
  if $FilteredMessages = \emptyset$ **then**
    if $|DiscussedAnalyses| > 1$ **then**
      return null
    else
      $FilteredMessages \leftarrow Messages$
    end if
  end if
  $NewNucleus \leftarrow \arg\max_{m \in FilteredMessages} m.fact.interestingness$
  $Messages.remove(NewNucleus)$
  return $NewNucleus$
end function

This formulation allows us to naturally produce both texts discussing a wide variety of different analyses, as well as focused texts about a single analysis. The behavior is driven by the system input: an input consisting of the results of multiple analytical tools naturally results in an overview-style text, whereas an input consisting of the outputs of only a single analytical tool results in an in-depth text.

We note that a thematic flow restriction, like the one used in satellite selection, could also be implemented in the nucleus selection procedure if the flow of the paragraphs turns out to be prohibitively incoherent. However, based on the observations of White [31] regarding the exchangeable order of the paragraphs in news reports (to which the reports being produced by the Reporter are similar in style), we do not expect this to be necessary, nor have we observed a need for such a modification based on our own experiments.

The planning then continues by selecting satellites for this nuclei, etc., until either a predefined maximum length, measured in paragraphs, is reached or there are no more sufficiently interesting nuclei to select. The maximal length is a tuneable hyperparameter which we set to 5 based on experimentation. In case of the nuclei, we calculate 'sufficiently' interesting as follows: the first nucleus of a document is always sufficiently interesting; the second nucleus is sufficient interesting if the score is at least 10% of the score of the first nucleus; and further nuclei are sufficiently interesting if they have scores greater than the score of the first nucleus multiplied by 0.3. In the alternative, any nucleus is sufficiently interesting if it has an interestingness value of at least 0.5. These values were obtained experimentally, and are tuneable hyperparameters that may be easily modified.

An evaluation of this document planning approach (in the context of news text generation showed statistically significantly improved performance over a simpler baseline method. The evaluation is described in more detail in [1], attached to this document as Appendix B.

The evaluated procedure differs slightly from that described above in that the news domain application contains additional "ancillary" messages that were known to be less related to the main theme of the document but still potentially inclusion worthy as, for example, points of comparison. The algorithm

	Our method			Baseline			
Statement	Median	Mean	SD.	Median	Mean	SD.	p_{MWU}
Q1 (1–7, ↑)	5	4.40	1.64	2	1.80	0.41	< 0.001*
Q2 (1–7, ↑)	5	4.33	1.76	2	1.60	0.51	< 0.001*
Q3 (1–7, ↓)	4	4.47	1.81	6	5.80	1.42	0.049
Q4 (1–7, ↓)	5	5.13	1.55	6	6.33	0.62	0.024
Q5 (1–5, 3 best)	3	2.93	0.59	4	4.07	0.70	< 0.001*

Table 4: Results obtained during the evaluation. Parentheses indicate answer ranges and whether the higher (↑), lower (↓) or middle values are to be interpreted as the best. The p_{MWU} column contains the (uncorrected) p-value of a two-sided Mann-Whitney U test. An asterisk indicates the p-value is statistically significant also after applying a Bonferroni correction to account for multiple tests.

inputs were modified so that the messages from this "*expanded set*" are only available as satellites. In addition, a "*set penalty*" factor is included in satellite scoring, increasingly penalizing the scores of satellite candidates belonging to the expanded as the distance to the nearest non-ancillary preceding message increases. The NewsEye procedure described above does not implement these features, as the NewsEye generation domain does not contain similar "ancillary" messages that would be delegated to the "expanded set".

The simpler baseline method, against which the method described above was evaluated, constructs paragraphs by always greedily selecting maximally important messages without regard for text coherence. It does not include the set penalty factor, but the ancillary messages are only available as satellites. The baseline method does not include any early stopping criteria, and as such always constructs the maximal allowed number of maximal length paragraphs provided that a sufficient number of messages is available.

During the evaluation, domain expert judges were shown texts generated by the general method described above (adapted for the news context) and the simpler baseline. After reading each text, the evaluators were asked to indicate their agreement with the following statements (translated from Finnish, the native language of the evaluators):

Q1: The text matches the heading
Q2: The text is coherent
Q3: The text lacks some pertinent information
Q4: The text contains unnecessary information
Q5: The text has a suitable length

For Q1–Q4, the judges indicated their agreement on a 7-point Likert scale ranging from 1 ('completely disagree') to 7 ('completely agree'). For Q5, the answers were provided on 5-point scale ranging from 1 ('clearly too short') to 3 ('length is suitable') to 5 ('clearly too long'). In addition, the judges were able to provide textual feedback for each individual text, as well as for the evaluation task as a whole. The judges' answers to Q1 – Q5, are aggregated in Table 4.

The results indicate that the employed method statistically significantly increases the document's coherence (Q2, mean 4.33 vs. 1.60, median 5 vs 2), the matching of the document's content to the document's theme (Q1, mean 4.40 vs. 1.80, median 5 vs 2), and produces documents of more suitable

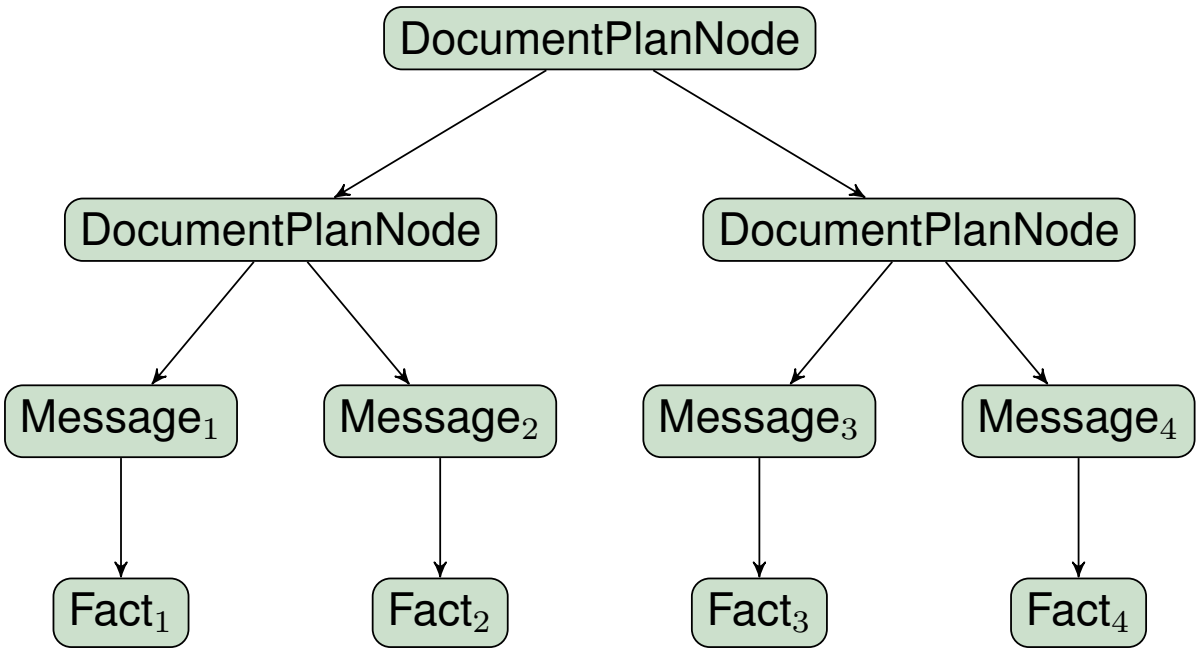

Figure 6: An abridged version of a Document Plan as produced by the Document Planner.

length (Q5, mean 2.93 vs. 4.07, median 3 vs 4, with 3 being best). The proposed method also seems to result in less unnecessary information being included in the document (Q4, mean 5.13 vs 6.33, median 5 vs 6), and in the text missing less necessary information (Q3, mean 4.47 vs 5.80, median 4 vs 6), but these effects are not statistically significant after correcting for multiple comparisons with the Bonferroni correction. The Bonferroni correction changes the threshold value used for deciding whether a result is statistically significant (usually $\alpha = 0.05$) by dividing it by the number of tests, n. Here, the new threshold for statistical significance is $\alpha_{bonf} = \frac{0.05}{5} = 0.01$. This addresses the multiple comparison problem which would result in an increase in false positive results as the number of statistical tests increases. Notably, as a very simple method of compensation, Bonferroni correction can "overcorrect", resulting in an increased amount of false negatives.

The output of the document planning procedure is a tree-structure, detailing the overall structure of the document: leafs correspond to Facts and the branches from the root node correspond to paragraphs. An abridged example of a document plan is provided in Figure 6. This document plan then acts as the input of the next processing phase of the pipeline.

5.3. Templates and Template Selection

As the next step in the NLG process, the language-independent messages need to be transformed to some type of linguistic constructs. In this version of the system, these language constructs correspond to individual phrases or short sentence-level templates.

These templates are provided to the system in a custom templating language. Figure 7 provides an excerpt of this templating language. As can be observed, a template group in our system consists of three parts:

1. A per-template language identifier such as '`en`' for English,

```
en: {result_key} appeared {result_value} times
en: {result_key} was found {result_value} times
| analysis_type = ExtractBigrams:Count
```

Figure 7: A hypothetical example of the templating language.

```
en: {result_key} had a relative count of {result_value}
fi: {result_key, case=gen} suhteellinen osuus oli {result_value}
de: {result_key} hatte eine relative Anzahl von {result_value}
fr: {result_key} ont un taux de comptage relatif de {result_value}
| analysis_type = ExtractBigrams:RelativeCount
```

Figure 8: An example of multilingual templates. The example also shows how information on grammatical case and other similar factors can be included in the slots.

.

2. a phrase (template) expressed in natural language with slots (indicated by curly brackets `{}`) that can be filled with information from the Fact data structures, and
3. conditions for using the templates in the group, such as `analysis_type = ExtractBigrams:Count`

Moreover, slots in the templates are optionally associated with grammatical cases and other additional information to instruct components further 'down' the pipeline on how to treat them. For example, a slot containing a numeric value can be augmented with the modifier `abs` (i.e. `{result_value, abs}`) to indicate that the number should be made into an absolute in a further processing stage. This allows us to, e.g. express a negative numeric change using the phrase '*decreased by 11.5*', rather than using the significantly more cumbersome phrasing '*had a change of -11.5*'.

Finally, square brackets in the templates (e.g. `[in {corpus}]`) can be used to indicate parts of the phrases that are optional in the sense that the phrase is still meaningful even without them. Templates with such optional segments are expanded into variants both with and without the optional segment.

The templating language was designed to allow for multilinguality in the system. Multilinguality is supported by allowing expressions in different languages to be specified within the same template group, i.e. by adding 'fi' or 'en' at the beginning of the template as shown in Figure 8. In many cases, adding new languages to the template group does not require creation or modification of the conditions of the group. It only requires the translation of the template text, as demonstrated by Figure 8. The aim of the templating language is that it makes it relatively simple for even non-technical domain experts to contribute to the creation of templates.

An important observation at this point is that the templates are, fundamentally, each related to a specific analytical tool. In other words, when a new tool is introduced, we need new associated templates. And when an analytical tool is removed from the larger system, the templates associated with it are no longer useful. This mapping between the templates and the analytical tools indicates that the templates can – and should – be organized on a per-tool basis. To this end, the templates are stored in per-tool databases (as shown in Figure 4) that are provided to the Reporter together with the per-tool message generators in joint components called 'Processor Resources'. These Processor Resources also contain (some of) the resources needed to lexicalize the templates, as discussed later in Section 5.4.

To select a suitable template for a given message in the document plan, the system first finds, for each fact, a template that can be used to express said fact. This is done by evaluating the conditional line against the messages. In the case that multiple valid templates exist for a certain fact, one is selected pseudo-randomly. This randomness is pseudo-random in the sense that the random number generator is re-initialized with a known constant starting position (a 'seed number') for every generation task. This means that every time the system is called with the same set of inputs it produces the same 'random' choices. This is useful in that it produces variety into the text for the human reader of the resulting report, but still makes the process deterministic for most relevant purposes insofar as development is concerned. This also means that if the same generation task is run multiple times, the same report is produced down to the minor linguistic choices.

Having identified a suitable pseudorandom template, a copy of found template is then attached to the tree as a child of each fact's parent message so that each slot of the template contains a link to the underlying fact. Individual slots rather than the whole template are associated with the facts, since the following aggregation phase can result in a single (modified) template containing slots referring to multiple facts. This phase results in a tree-like structure such as the one shown in Figure 9.

In the case where the system is unable to locate a template that can express a certain fact, the system simply removes the fact from the document. We note that this behaviour is only a contingency, and we are not aware of any cases where this would happen in the present version of the system. Finally, the resulting document plan (with added templates) is provided to the lexicalization component for further processing.

5.4. Lexicalization

As show in Figure 9, the templates attached and filled by the previous stage of the pipeline still contain unlexicalized content in the slots of the templates. These unlexicalized segments most commonly come in the form of references to entities, such as `[ENTITY:NEWSPAPER:arbeiter_zeitung]` in Figure 9. These are handled by a Referring Expression Generator component in a future step. In some cases, however, the values referred to by the slots in the templates are tags, such as `[PUB_YEAR:1936]` in the same example.

To this end, the system allows for inclusion of *lexical resolvers*. In the aforementioned case of the tag `[PUB_YEAR:1936]`, a specific lexical resolver matches the general format of `[PUB_YEAR:X]`. It then extracts the value in place of the X (here, the year), and replaces that value into a token of its own, replacing the original tag.

These lexical resolvers are not limited to just working with tags, but can be applied to transform any token into one or more tokens. The resolvers are applied iteratively until the sentences stabilize, meaning that a lexical resolver's output can be processed further by another lexical resolver. They thus give a significant increase in the expressive power of the system. This power, however, comes at the cost of a significantly more complex syntax – most of the resolvers do their matching via regular expressions – and are thus not a panacea that would also replace the templates.

A more complex lexicalization challenge is presented by the `TopicModelDocumentLinking` process, which effectively produces lists of documents. For example, one of the English language templates

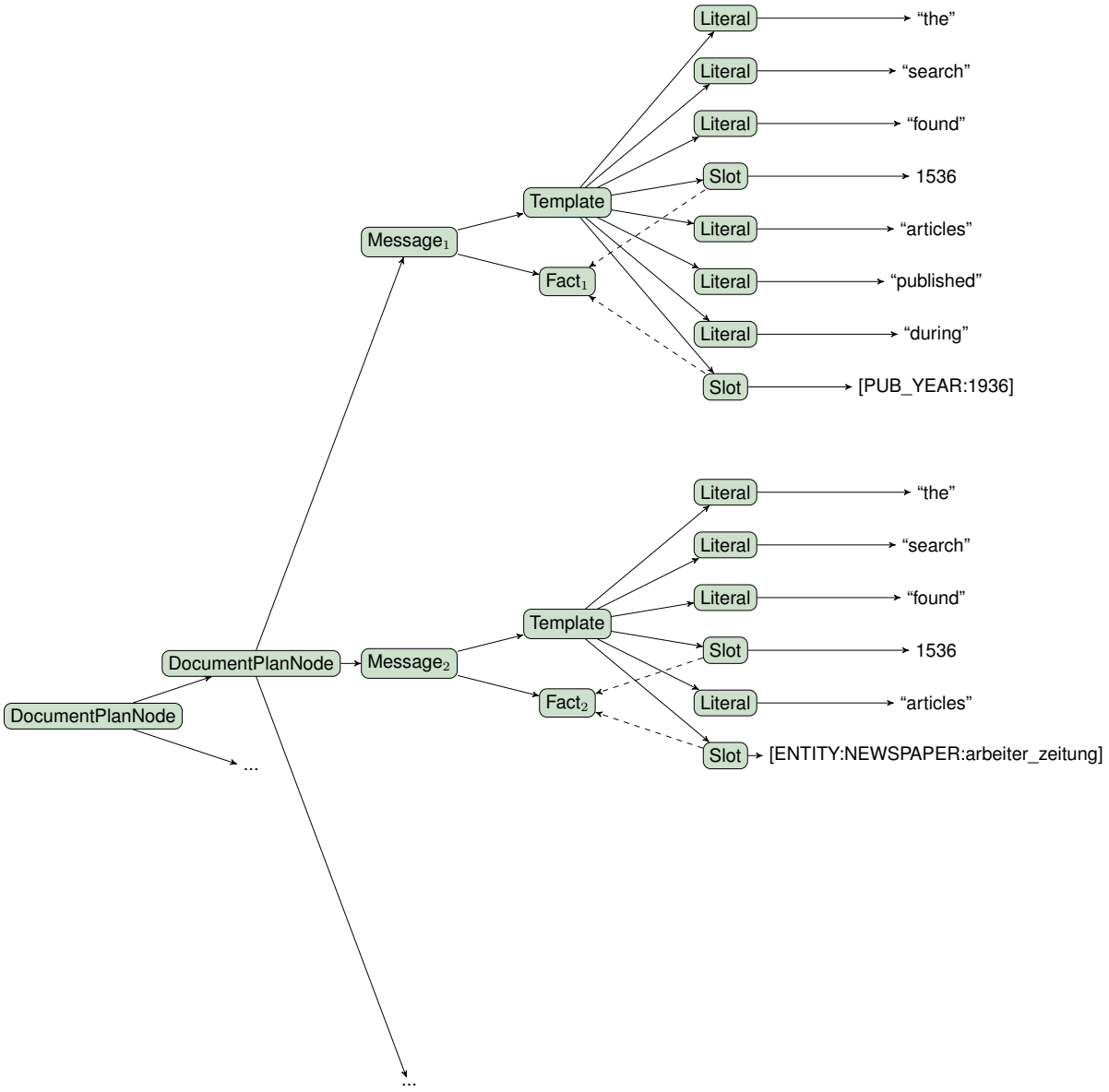

Figure 9: A Document Plan after attaching Templates.

used with this processor is

based on topic modelling, the following similar articles were identified: {result_value}

In the associated message, the `result_value` field contains an expression of the following format:

`[LinkedArticleList:11|21|234|32]`

Notably, the amount of |-separated components is not known *apriori*. For such scenarios, the Reporter has the ability to parse the |-separated components so that each is realized as a separate component. The tooling required for this parsing also enables us to easily define what separators are to be used for the list elements. This allows us to realize

`[LinkedArticleList:11|21|234|32]`

as, in the case of English,

`[LinkedArticle:11], [LinkedArticle:21], [LinkedArticle:234] and [LinkedArticle:32]`

By then using another lexical resolver to realize `[LinkedArticle:11]` as, for example, '#11', we can produce an expression such as

based on topic modelling, the following similar articles were identified: #11, #21, #234, #32

The above example is slightly simplified, in that the real article identifiers are more complex expressions such as '`arbeiter_zeitung_aze19200124_article_79`'. While more complex, they also allow for a basic understanding of what newspaper and from what date the article is from.

The lexicalization process is highly dependent on the details of message generation (rather, the individual result parsers – see Section 5.1.3). For each slotted value in a template, the lexicalization process must be able to produce a natural language expression that describes said slotted value. As the concrete values present in the Facts are decided by the individual result parsers, corresponding per-analysis logic is needed during lexicalization. These are integrated to the system similarly to the result parsers, and are show in Figure 4 as the small boxes labeled 'Lexical Resolvers'. As they are associated with result parsers, and thus individual analytical tools, they too are included in Processor Resources.

After lexicalization, the document plan is a tree akin to that shown in Figure 10. This modified document plan is then passed to the next component in the pipeline.

5.5. Aggregation

In an effort to create richer and more fluent sentences, the Reporter contains an aggregation component that can be used to combine individual phrases to more complex phrases that contain less duplicate information. Aggregation is a notoriously complex and delicate process and contains significant danger for misleading the reader. Our simple aggregation component aggregates two consecutive templates if they contain a common prefix of one or more tokens. In such cases, the shared prefix in the latter phrase is replaced by a conjunction such as 'and' in English.

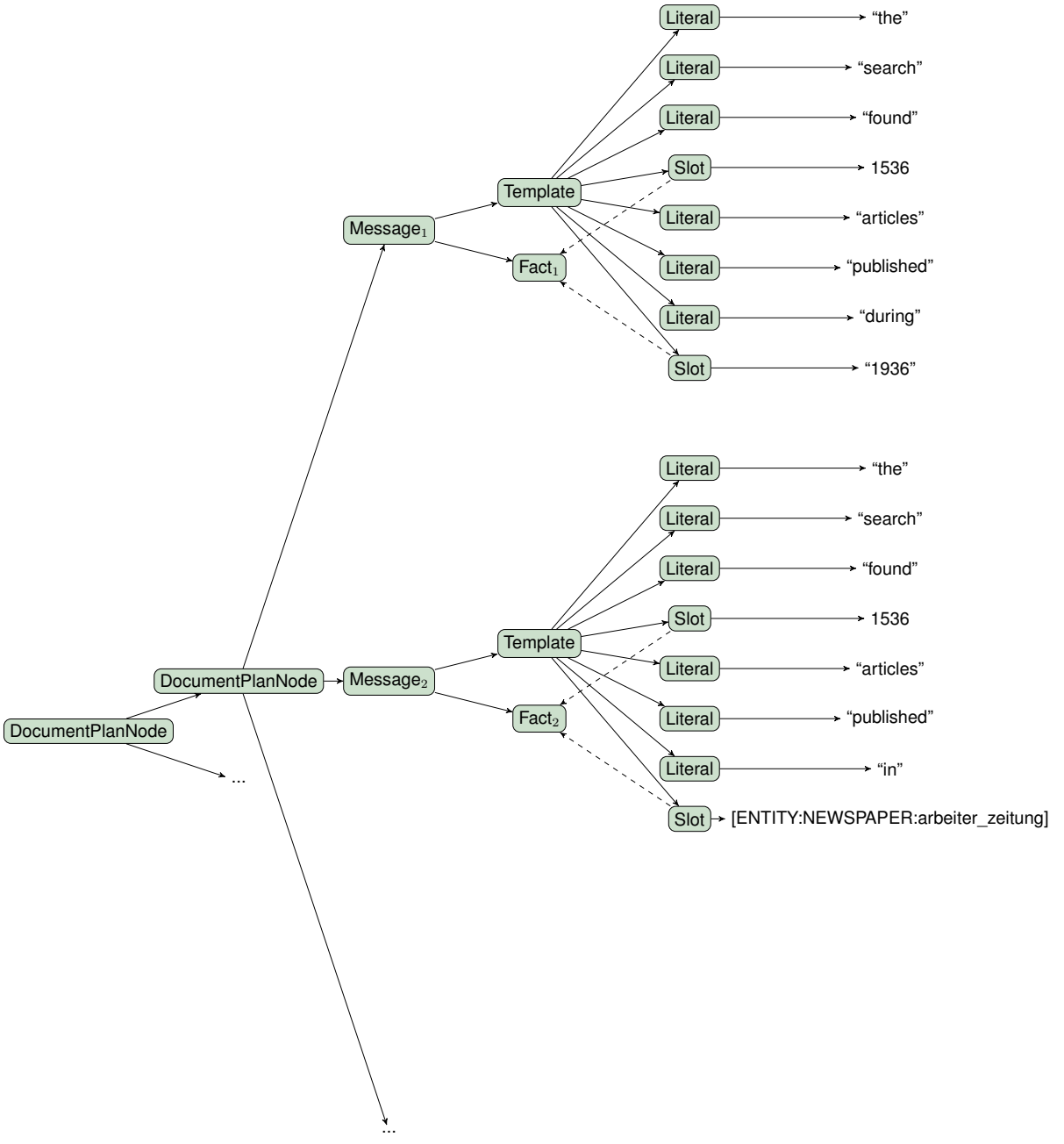

Figure 10: Part of a Document Plan after Lexicalization.

For example, using a natural language example, consider the phrases "***The stem pair "president harding"*** *appeared 7 times*" and "***The stem pair "president harding"*** *had a relative count of 0.0293*". Here, we indicate a shared prefix in bold. Intuitively, these phrases can be aggregated to "***The stem pair "president harding"*** **appeared 7 times and had a relative count of 0.0293**". While the result shown in this specific example could be achieved by considering the words of the phrases only, that method has two significant issues.

The first issue is the aggregation of numbers as part of the shared prefix. For example, we wish to explicitly disallow aggregation of "*The stem pair "president harding" appeared 7 times in 1915*" and "*The stem pair "president harding" appeared 7 times in 1914*" as "*The stem pair "president harding" appeared 7 times in 1915 and 1914*" which could be interpreted as meaning that the phrase appeared a total of 7 in 1915 and 1914 together. That is, that the phrase appeared, for example, 3 times in 1914 and 4 times in 1915. As such, to avoid misleading the reader, we conduct aggregation so that the shared prefix can not include slots referencing `result_value` fields, even if the contents of those fields are identical. The resulting aggregation would thus be "*The stem pair "president harding" appeared 7 times in 1915 and 7 times in 1914*." While longer, this sentence is not ambiguous in the same way as the problematic aggregation above.

Another problem is presented by cases wherein the aggregator identifies a part of a larger phrase as belonging to the shared prefix. We chiefly observed this as occurring as aggregation of phrases such as "*the most positive value was 15*" and "*the most negative value was -1*" as "*the most positive value was 15 and negative value was -1*". Here, the resulting aggregated phrase is ungrammatical. To address this problem, we further limit the aggregation so that the aggregator only considers a shared prefix to the point wherein the last element of a shared prefix is some slot.

As a special case, we allow prefixes where the prefix itself contains no slot, but both component sentences immediately follow the prefix with a `result_value` field. This rule is used, for example, to aggregate the phrases "***the search found*** *1536 articles published during 1936*" and "***the search found*** *1536 articles published in 'Arbeiter Zeitung'*". as "***the search found*** *1536 artiches published during 1936 and 1536 articles published in 'Arbeiter Zeitung'*". Here, the bolded parts are Literals, but aggregation is allowed as the joint prefix is followed in both source sentences by a `result_value` field, realized in the sentences as '1536' in both cases.

The resulting aggregation system is not perfect, but we believe it strikes a suitable balance between level of aggregation, technical complexity and requirement for per-template metadata. Furthermore, due to the pipeline nature of the Reporter, the Aggregator can be trivially disabled if the approach is later identified as being unsuitable. In that case, the resulting texts will be longer and contain more repetition, but will also be more clear on the level of individual sentences.

The result of the aggregation phase is, again, a modified document plan such as the one shown in Figure 11. This plan is then passed to the next component in the pipeline.

5.6. Referring Expression Generation

Following aggregation, the document plan still contains some unlexicalized content in form of references to entities such as newspapers, languages and dates. These are lexicalized as part of the Referring Expression Generation process. This process is distinct from the lexicalization process in two ways: first, lexicalizing a newspaper's name or the name of a month is not related to a specific analysis,

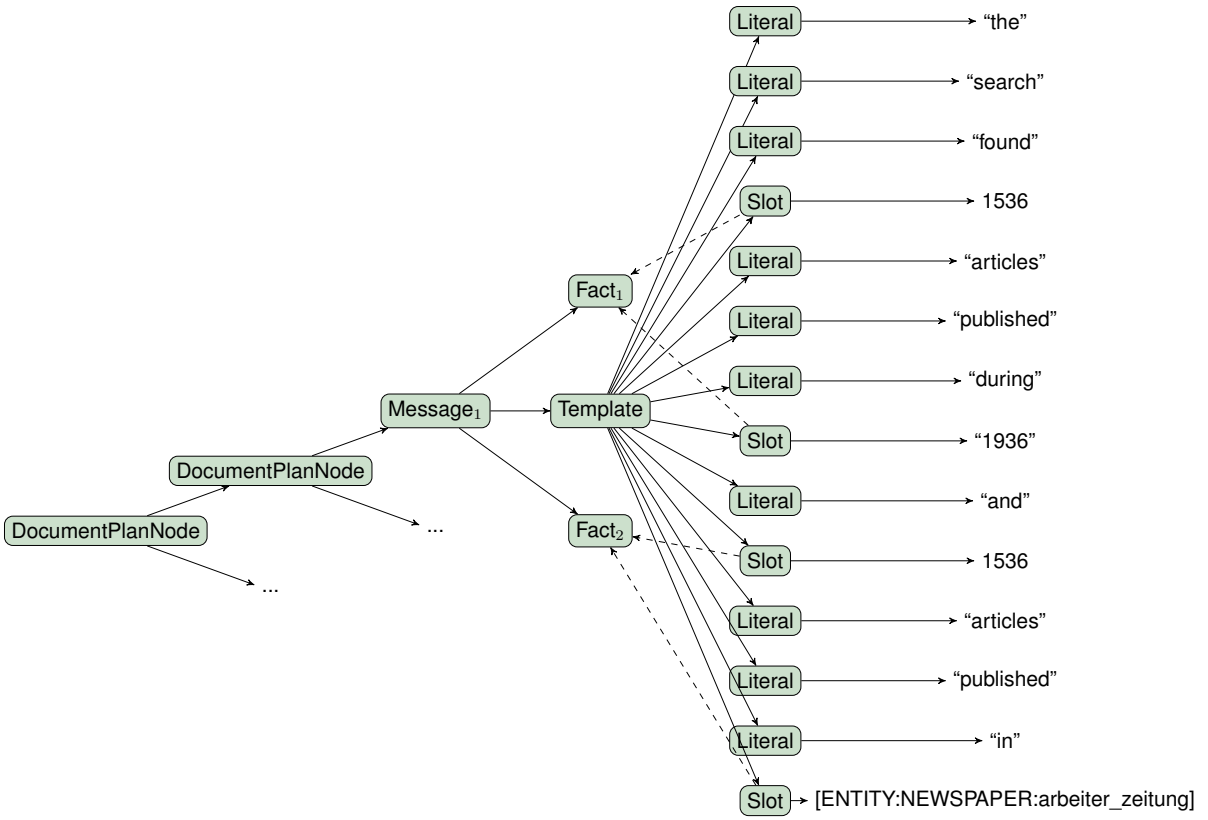

Figure 11: Part of a Document Plan after aggregation.

meaning that this step should not be aligned to any specific Processor Resource in the same way as the Lexicalization resources were. Second, in some cases we might wish to replace e.g. the name of a newspaper with the word 'it', provided that the surrounding sentence makes it clear what is being referred to.

During this step, for example, the tag '`[ENTITY:NEWSPAPER:arbeiter_zeitung]`' would get replaced with the phrase '*Arbeiter Zeitung*', and the tag '`[TIME:between_years:1915:1916]`' would be realized as 'between 1915 and 1916'. In improving this component of the pipeline from the previous iteration, we observed that the date realization code was both relatively complex and separable from the code used to realize references to entities. As such, we separated these two parts into distinct modules, as shown in Figure 4.

The Referring Expression Generation module, at the present, produces referring expressions for languages, newspapers and general entities (e.g. locations and people). The module is equipped to use shorter expressions, such as 'it', in place of the full expressions (e.g. 'Arbeiter Zeitung'), but our initial trials indicated that such a feature was not improving the quality of the text significantly, but occasionally caused confusing and unclear referential structures. As such, we decided to not enable this feature. A less-complex, but conceptually similar, feature used in date reference generation allows the use of expressions such as 'the same year' in place of year numbers, but such expressions seldom used.

The output of the Referring Expression Generation module is, finally, the fully lexicalized document plan as shown in Figure 12. This plan, however, still needs additional post-processing and is thus passed on to the next component.

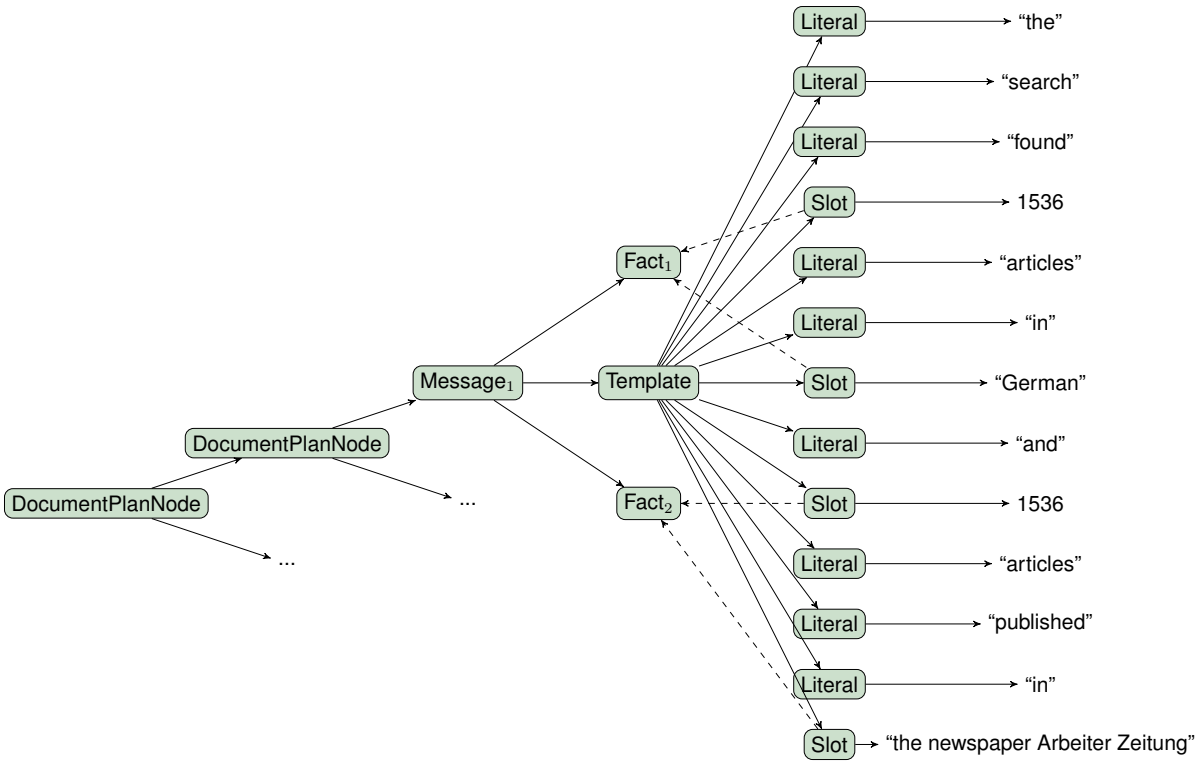

Figure 12: An abridged version of a Document Plan after Referring Expression Generation.

5.7. Morphological Realization

While the English language running example we have been using is very close to correct language, the processing pipeline is not yet complete. Especially in cases of more complex morphology, certain words in the document plan still need to be inflected into their right morphological forms. In English, this is relatively straightforward. However, in other languages such as Finnish, the morphological realization process is significantly more complex.

During morphological realization, each token of the document plan is inspected individually. If it contains morphological information, such as a 'case' attribute, the token and the case are handed to a language specific morphological realizer (shown in Figure 4) which correctly inflects the token.

As the morphological realization process is extremely complex for languages such as Finnish, we use 3rd party libraries for this task. At the present, we have implemented realization systems for both English and Finnish using the UralicNLP library [32]. We have not found it necessary to include 3rd party morphological realization libraries for the French and German languages, but the design of the systems allows them to be added easily if they become necessary in the future. The English module is also, in reality, redundant in that none of the templates require its use.

5.8. Realization into HTML

Finally, the document plan, which has been modified by all the previous components, is given to the surface realizer to be realized into a format that can be displayed to the end user. As seen in Figure 12, the document plan at the onset of surface realization is in a state where natural language expressions

The search found 147610 articles in Finnish and 147610 articles from the newspaper Uusi suometar. The search found 9570 articles published during 1913 and 9114 articles published during 1916. The search found 8943 articles published during 1914.

Between 1869 and 1918, the largest yearly amount of relevant articles in Uusi suometar was 9570 and smallest non-zero yearly amount of relevant articles in Uusi suometar was 116. Between 1869 and 1918, the average amount of relevant articles in Uusi suometar was 2952.2. 116 relevant articles were found in issues of Uusi suometar published during the year 1869. 225 relevant articles were found in issues of Uusi suometar published during the year 1870.

The most prominent entities in the corpus were: Helsinki (salience = 0.73, stance = 2.0743014789769545e-05), Helsinki Airport (salience = 0.024, stance = 0) and Hanko railway station (salience = 0.011, stance = 0).

Tallinn was discussed during 17 distinct years between 1889 and 1918. The mean sentiments towards Tallinn between 1889 and 1918 was 0.059. The most negative sentiment towards Tallinn (0) occurred at 1889. The most positive sentiment towards Tallinn (1) occurred at 1894. Helsinki was discussed during 59 distinct years between 1852 and 1918

Figure 13: Two excerpts from reports produced by the Reporter.

can be formed by traversing the leaves of the tree. The higher-in-the-tree nodes of the document plan are used to distinguish limits of paragraph.

The surface realizer also adds typographical details such as capitalizing the first word of each sentence, adding sentence-final punctuation and removing extranous whitespace that sometimes remains in the text as an artefact of the previous generation stages.

The final stage of the surface realization is the production of a flat text representation which incorporates any necessary Hypertext Markup Language (HTML) tags. Currently, the Reporter can produce three types of different HTML structures: text paragraphs such as found in standard text documents, list structures where the individual sentences are presented as bullet points and enumerated lists.

Excerpts from the resulting texts are shown in Figure 13.

5.9. Link Generation

The reports produced by the Reporter are in almost all cases ‘highlights’ of some larger, underlying, analytical result. For example, while the natural language results might only describe the yearly article counts for the three – most interesting in the view of the Investigator – years, the underlying data contains the same information for all the years present in the dataset being analysed. As such, it is useful to be able to link from the textual descriptions to either a non-linguistic representation of the underlying analytical results, or perhaps another report that discusses said result in more detail.

While the Reporter is unable to construct such links directly, as it is agnostic to the user interface being used to display the result, it provides the necessary information to conduct this linking in the user interface as a post-processing step. This is achieved by embedding in every template a special tag of the form `[LINK:210811b4-42fa-499e-931e-3d89906a51b6]`. Here, the post-colon section of the tag is a UUID uniquely identifying what analysis the information presented in the sentence originates from. Similar links are also generated to point to articles discussed in the reports. These tags can be easily parsed in the UI to form suitable links to the larger results of the relevant analysis.

As such links are not always useful, the Reporter's API (see Section 7) allows an optional `links` flag to be included in the report generation requests. While the links are always generated (for simplicity of the generation process), both omitting the flag and setting it to `false` result in a post-processing step removing them from the reports. The decision to treat the omission of the flag as a negative value was based on a need to keep the API backwards compatible with the original API.

6. Header Generation and Multi-part reports

The Reporter also generates a header or a title for each document it produces. The generation of the title uses the same method as described above with three modifications. The first of these is that the document plan is limited to a single fact. This ensures that the resulting document can only contain a single template.

Second, the template selection is conducted from among a separate group of header templates that describe the `corpus` field of the selected message. These are provided as a separate module comprising the header templates and their lexical resolvers. The message generation is handled by the standard result parsers. The header might be, for example, '*Analysis of the dataset 'myDataset' filtered by the query "femme"*'.

Finally, the surface realizer wraps the 'document' (i.e. the single sentence) in the `<h1></h1>` HTML tags that indicate the text is a header.

As might be inferred from the above description of the header generation, the Reporter pipeline assumes that all the results discussed under a single heading discuss a single dataset. The Reporter makes no attempt to describe in the individual sentences and paragraphs which dataset they are describing. At the same time, more complex investigations by the Investigator provide results about multiple, slightly different datasets as well as results that compare datasets to each other.

In such cases, the Reporter splits the input into multiple generation tasks so that each task is about a specific, individual, dataset. These reports – and their headlines, generated as described above – are then concatenated together into one larger document. The headers thus turn into section headers, with the header describing the dataset that is discussed in each section.

This process can naturally lead to an unwieldy amount of 'sub-reports' in cases where the inputs contains information about a very large amount of corpora. To prevent excessively long reports, the reporter seeks to determine both the most salient sub-reports to include in the larger report, as well as a suitable ordering for the sub-reports.

As a proxy for sub-report salience, the Reporter uses the maximal interestingness of any of the individual messages in a sub-report. An alternative approach would be, e.g. the mean interestingness, but this would penalize sub-reports that contain one or two highly interesting facts together with a plethora of medium-to-low interestingness facts. Using this salience proxy, the top-5 most salient sub-reports are selected for inclusion in the final report.

This final report is then constructed so that the sub-reports appear in a decreasing order of salience, but with sub-reports discussing the same dataset grouped together. That is, if both the most and least salient sub-report (that make the top-5 cut) are about the same dataset, the least salient sub-report is lifted so that it immediately follows the most salient sub-report. This prevents behaviour where the overall report starts by discussing some specific dataset, then changes topic to another dataset, and finally returns to the original dataset.

7. Integration with the Personal Research Assistant

The Reporter is integrated to the other NewsEye components in two ways. First, it communicates with the Controller of the Personal Research Assistant, and through it, the Investigator. Second, the Reporter contains functionalities that enable it to report on the results of each individual analytical tool used by the Investigator.

7.1. Integration with Assistant Control

We have described in Section 2 the relation of the NewsEye Reporter (i.e. the system described in this Document) to the rest of the NewsEye Personal Research Assistant (Work Package 5), the NewsEye User Interface (Work Package 7) and other components produced by Work Packages 3 and 4 in broad terms. In practice, to enable the Assistant Controller to start a generation task, the Reporter offers a simple Application Programming Interface (API) with four endpoints.

The first endpoint, `/api/languages`, simply reports the languages supported by the Reporter. This enables the technical users (such as the NewsEye User Interface) to dynamically display a list of available reporting languages to the end user. The second endpoint, `/api/formats`, returns the set of valid body formatting options. The third endpoint, `/api/report` is used to generate new reports based on the analysis provided as a parameter. The `/api/report` endpoint expects the input to be encoded in form data format. As this is not always simple, we also provide an alternative, fourth, endpoint `/api/report/json` that provides the same service as the `/api/report` endpoint, but expects the input to be provided as a JSON document. All API endpoints produce responses in the industry standard format, JSON. The Reporter's API is described in detail, with example responses, in Appendix A.

7.2. Integration with Individual Analytical Tools

While the Reporter is not directly connected to the individual tools used by the Investigator, it nevertheless needs some tailoring for each such tool. Without this tailoring, the Reporter cannot know how to report on the analytical results provided by the tools. As described in various parts of Section 5, this tailoring needs to occur on three different levels: message generation, templates and lexicalization.

First, the message generator (see Section 5.1.3) needs to be tailored for each type of analytical results so that the results can be correctly translated to the fact data structures (see Section 5.1.1). This tailoring is provided in the format of *Python 3* code implementing the relevant result parsers.

Second, it must be ensured that each type of analytical result has a set of templates (see Section 5.3) that allow them to be expressed in natural language. These templates need to be provided in the templating language employed by the Reporter.

Third, the lexicalization component (see Section 5.4) must be tailored by providing any needed lexical resolvers, so that all concepts embedded into the templates can be expressed in natural language. Like the message generator tailoring, this tailoring is also provided as Python 3 code.

All three elements are provided to the Reporter as a single Python 3 module, the entry point to which is a class that implements a pre-specified interface. This class provides access to the templates (as a multiline string), the result parser (as a callable method with a specified signature) and to the lexical resolvers (as another list of Python 3 classes that all implement a specified interface). This is denoted as the `Processor Resource` box in Figure 4. The resource is then integrated into the rest of the Reporter in a very simple manner, requiring only two lines of code.

8. Summarization

Text Summarization (TS) aims to create a summary containing the main ideas of a textual document. TS systems can be extractive, compressive or abstractive. Extractive methods estimate the relevance of sentences in a document to generate a summary by concatenating the most relevant sentences. Compressive methods compress sentences to reduce the length of sentences and to preserve only the main information. Then, they generate summaries by concatenating the most relevant sentences and compressions of a document. Finally, abstractive methods analyze a document and generate a summary with new sentences that contain the meaning of the source documents [33].

Many of the state-of-the-art methods for TS are of the extractive class. Recent systems have used compressive and abstractive approaches [34] to improve the informativeness and the grammatical quality of summaries. However, these approaches require specific resources for each language or external data and combination of different methods that limit the adaptability of these methods to generate summaries into other languages. Recently, transformer architectures have reached the state of the art in several downstream NLP tasks by pre-forming models on large generic corpora and then adapting them to specific tasks with high performance.

We have therefore focused on two extractive TS approaches that generate stable results and can be easily adapted to any language and two transformer-based systems for English documents. More precisely, we describe two well known state-of-the-art TS systems (TextRank [35] and MMR [36]), and an extractive and abstractive systems based on the transformer architecture.

8.1. Text summarization systems

TextRank is a graph-based approach that was originally devised to estimate the relevance of pages from the number of citations or the study of the Web structure [35]. This approach makes decisions about the importance of a vertex based on the global information coming from the recursive analysis

of the complete graph. In the scope of automatic summarization, it is observed that the document is represented by a graph of textual units (sentences) connected through relations resulting from similarity calculations. The sentences are then selected according to the criteria of centrality or prestige in the graph where each sentence in represented for a vertex V_i, and grouped to produce extracts of the text.

$$S(V_i) = (1-d) + d \times \sum_{j \in In(V_i)} \frac{S(V_j)}{|Out(V_j)|} \tag{2}$$

where V_i is a vertex of the graph, $In(V_i)$ are the incoming links to the vertex V_i, $Out(V_i)$ are the outgoing links to the vertex V_i, d is a damping factor that can be set between 0 and 1, which has the role of integrating into the model the probability of jumping from given vertex to another random vertex in the graph. Finally, $S(V_i)$ is the summarization score of V_i. All $S(V_i)$ are randomly initialized and updated until convergence.

Maximal Marginal Relevance (MMR) approach produces a summary based on the relevance and the redundancy of the sentences [36]. It strives to reduce redundancy while maintaining query relevance in re-ranking retrieved documents and in selecting appropriate passages for text summarization. They use a linear combination to measure the relevance and novelty independently:

$$MMR = \arg\max_{s \in D \setminus S} [\lambda \times sim_1(S+s, D) - (1-\lambda) \times sim_2(S, D)] \tag{3}$$

where S is the summary, D is the document, s is the analysed sentence to be added to the summary S, $D \setminus S$ is the set difference, and the functions sim_1 and sim_2 are similarity measures. We follow the standard strategy that consist to initialize S to $\emptyset$ and calculated the similarity measures as the cosine of the TF-IDF representations.

Finally, we analyse two BERT-based systems. The first one uses the BERT model for text embeddings and K-Means clustering to identify sentences closest to the centroid for summary selection [37]. This approach generates summaries by extracting the sentences that have the most similar content to the main topic of the document. The second method produces abstractive summaries by using the BART model to analyse documents and generate summaries [38]. The BART model was finetuned on CNN-DailyMail dataset that contains over 280,000 document-summaries pairs.

8.2. Datasets

We performed the tests using the MultiLing Pilot 2011 dataset [39]. This dataset contains documents of news topics taken from the WikiNews[1] website in several language versions. These documents mainly focus on news which may be described or happen in different moments in time. Each language version is composed of 10 topics, each topic having 10 source texts and 3 reference summaries. Each reference summary contains a maximum of 250 words. Specifically, we used the English and French language versions to evaluate our baselines.

[1] https://www.wikinews.org/

8.3. Automatic evaluation

TS approaches are evaluated based on the informativeness of their summaries. As references are assumed to contain the key information, we calculated informativeness scores counting the word n-grams in common between the system output and the reference summaries. The ROUGE measure developed by Lin [40] compares the differences between the distribution of words of the candidate summary and a set of reference summaries. The comparison is made splitting into n-grams both the candidate and the reference to calculate their intersection. Standard n-gram values for ROUGE are 1-gram and 2-gram, both expressed as:

$$\text{ROUGE} - n = \frac{\sum_{Sum_{ref} \in \mathcal{S}} \sum_{n-gram \in Sum_{ref}} Count_{Sum_{can}}(n-gram)}{\sum_{Sum_{ref} \in \mathcal{S}} \sum_{n-grams \in Sum_{ref}} Count_{Sum_{ref}}(n-gram)}, \tag{4}$$

where n is the n-gram order, $\mathcal{S}$ is the set of all reference summaries, Sum_{can} the candidate summary, Sum_{ref} the reference summary, and $Count_{Sum_x}(n-gram)$ is the number of occurrences of the $n-gram$ in the reference summary (when $x = ref$) or in the candidate summary (when $x = can$). A third common ROUGE-n variation is ROUGE-SUγ. This ROUGE-n variation takes into account skip units (SU) $\leq \gamma$ allowing some arbitrary gaps but in a controlled fashion. For each ROUGE measure, we analyse the precision (P), recall (R) and F-measure (F1).

8.4. Experimental setup

All systems concatenate all documents of a topic and analyse them as a single document. Then, they generate summaries composed of 250 words with the most relevant sentences, while the redundant sentences are discarded. We then apply the cosine similarity as defined in Equation 5.

$$cos_{sim}(s_i, s_j) = \frac{< \mathbf{s}_i, \mathbf{s}_j >}{|\mathbf{s}_i| * |\mathbf{s}_j|} \tag{5}$$

where $\mathbf{s}_x$ is the vector representation of the sentence s_x and $< \cdot, \cdot >$ is the dot product operation. A threshold of 0.5[2] was set to remove redundant sentences in the extractive summary generation. We used the python library BERT-EXTRACTIVE-SUMMARIZER[3] and the HuggingFace library[4] for extractive and abstractive Transformer summaries, respectively.

As we have only found pre-trained models for the summarization of English documents, we only used the transformer-based systems on the English dataset.

8.5. Experimental evaluation

The results of the automatic evaluation using the MultiLing Pilot 2011 datasets are described in Tables 5 and 6. For both languages, TextRank outperformed MMR and transformer-based systems for all ROUGE measures. Indeed, TextRank analyses how the similarity between sentences in all documents impacts in the informativeness of the documents. This strategy enables it to identify which sentences contains the main information and selects these sentences to generate the summary.

[2] This value was empirically chosen.
[3] https://pypi.org/project/bert-extractive-summarizer/
[4] https://github.com/huggingface/transformers

MMR generates a summary based on the analysis of novelty and redundancy of sentences. However, this method does not take into account the impact of the information contained in one sentence on the overall information contained in all documents, which limited the quality of its summaries.

Although transformer-based models generate a more in-depth analysis than the two baselines, they did not achieve the best results. One reason may be the limited input size for transformer-based models, which is limited to 1024 tokens in the original architecture [37]. This limitation reduces the amount of content that is analyzed by the model and, as a result, causes the generation of poor summaries.

	ROUGE-1			ROUGE-2			ROUGE-SU		
	P	R	F1	P	R	F1	P	R	F1
MMR	0.4521	0.4517	0.4519	0.1345	0.1342	0.1343	0.1937	0.1934	0.1935
TextRank	**0.4851**	**0.4769**	**0.4809**	**0.1565**	**0.1538**	**0.1551**	**0.2118**	**0.2049**	**0.2082**
Transf.-ext.	0.4134	0.4093	0.4113	0.1071	0.1059	0.1065	0.1574	0.1542	0.1557
Transf.-abst.	0.3059	0.4378	0.3600	0.0739	0.1056	0.0869	0.0903	0.1837	0.1209

Table 5: ROUGE results for the English version of the MultiLing Pilot 2011 dataset.

	ROUGE-1			ROUGE-2			ROUGE-SU		
	P	R	F1	P	R	F1	P	R	F1
MMR	0.4691	0.4722	0.4705	0.1535	0.1548	0.1541	0.2151	0.2184	0.2164
TextRank	**0.4874**	**0.4861**	**0.4866**	**0.1610**	**0.1600**	**0.1604**	**0.2250**	**0.2230**	**0.2239**

Table 6: ROUGE results for the French version of the MultiLing Pilot 2011 dataset.

8.6. Integration to Reporter

As the design philosophy of the Reporter (and the NewsEye Personal Research Assistant as a whole) does not allow for a summarization module to be embedded into the Reporter, we instead chose to implement the summarization aspect of report generation as an additional processor employed by the Investigator as part of it's experimentation with a dataset. The summarization processor is then integrated to the Reporter as any other processor, with a processor resource consisting of a template database, a result parser and relevant lexical realizers.

This method of integration also has the benefit of enabling future developments to modify the summarization component (either by fine-tuning or completely replacing it) without necessitating any changes in to the Reporter.

9. Evaluation

In evaluating the Reporter, we must consider two distinct aspects. The first of these is how well the Reporter matches the needs of its users. A technically marvelous system, while academically interesting, should not be considered a success if it was completely useless to its purported users. On the other hand, we must also evaluate the system in terms of whether it accomplishes its non-user-centric design goals, such as requirements for extensibility. We will consider these aspects starting from the non-user-centric design goals identified in Section 4. This will be followed by an evaluation of the fitness for purpose based on user testing sessions conducted with digital humanities domain experts.

9.1. Technical evaluation

As identified in Section 4, a fundamental requirement of the Reporter is that its output should always be *correct* in relation to the results identified by the Investigator.[5] By implementing the Reporter as a modular rule-based pipeline architecture, we avoid the hallucination issue commonly observed in neural NLG systems. Hallucination refers to the NLG system producing imaginary output rather than sticking to the given facts only.

While quantifying the 'correctness' of an NLG system is difficult – only in the last year have we seen the first academic works attempting to formalize methods for evaluating accuracy in texts that are not extremely short [41] – we are not aware of any cases wherein the Reporter produces output not consistent with the system input. Any Reporter-caused problems identified in previous versions of the system (e.g. misleading output due to aggregation, as discussed in Section 5.5) have been addressed. Furthermore, the rule-based modular nature of the Reporter allows for surgical corrections to address any hypothetical correctness issues if identified later-on. As such, we believe that the Reporter fulfills its design goals with regard to correctness.

The second requirement identified in Section 4 is *extensibility*. Optimally, the Reporter should be easily extensible with regard to both new languages and new analytical tools. With regard to extensibility of analytical tools, the reporter was constructed in an iterative manner: the analysis-specific Processor Resources were implemented and introduced to the system one-by-one. Thus, the process of creating the Reporter, itself, was a simulation of how feasible it is to extend the system with new analyses. During such an extension, time was principally spent on two tasks: understanding the input payloads on the level required to parse it into meaningful messages and in sourcing templates and lexical resolvers for all the system languages. In the case of the latter, we found it sufficient to construct full natural language example sentences that were then translated by domain experts that speak the target language. By a well-designed selection of the example sentences and their partial overlaps, it was possible for technical staff who did not speak the languages themselves to construct both the templates and the lexical resolvers. Naturally, this process is significantly simplified wherein the technical staff is directly able to contribute the templates and the lexical resolvers. In general, we believe that the Reporter fulfills the requirement for analytical extensibility very well.

With regard to extensiblity to new languages, we similarly simulated the extension process as part of the system construction: the Reporter was first implemented in English and Finnish, and support for both German and French was added only at the end of the development effort. Based on our experience, extending the system to a completely new language is the simplest when the language is morphologically simple and the extensions is done by a native speaker of the language. In case of morphologically complex languages, an increased amount of developer time is needed to implement the relevant morphological realizer. For morphologically simple languages, such as English, such a realizer might not even be needed. When the system is being extended to a language not spoken natively by the people in charge of the extension, further time is also needed to source the necessary translations from domain expert native speakers of the new language. Based on our experiences in extending the initial system from English to first Finnish and then German and French, we believe the system is easily

[5] This requirement's importance is exemplified by the feedback of internal domain expert testers, who observed that due to a programming fault the Reporter was referring to topics from a topic model with numbers that were off by one, i.e., topic #5 was called 'topic #4'. While they were able to identify and thus mentally adjust for the problem, it made the testers suspicious of all the other information contained in the results.

extensible at least within the scope of the majority European languages. As such, we believe the overall extensibility of the system – with regard to both languages and analyses – is good.

In terms of system *explanability* – the third requirement identified in Section 4 – our use of rule-based processing has resulted in a system that can be inspected in detail to precisely identify why the system took any specific action. At no stage does the system employ methods such as neural networks that would introduce black-box processes that would prohibit inspection or introspection. This has been demonstrated by our ability to identify and surgically correct the causes of problems identified by domain expert testers. As such, we believe this requirement is well fulfilled by the Reporter.

The final requirement identified in Section 4 is multilinguality, namely a requirement that the system be able to produce text in multiple languages. We believe the ability of the system to produce text in four different languages is by itself sufficient evidence to show that this requirement is well covered. By including Finnish as one of the languages supported by the system, we also demonstrate that the multilinguality is not only restricted to Indo-European languages. At the same time, while we are not aware of any examples, we cannot overrule the possibility that some of the fundamental assumptions behind the system's design were incompatible with some other language we are not familiar with. In any case, we conclude that the system is at the minimum highly multilingual within the sphere of European languages.

9.2. User-centric evaluation

While the above technical considerations are important, they are fundamentally secondary to the fitness of the Reporter for its intended audience's needs: no matter how correct, extensible, explainable and multilingual the Reporter is, it can only be viewed as a success if it is useful to the historians it is intended to be used by. In our view, this fitness for purpose has two primary aspects. First, the information presented in the reports must be both relevant to the user and presented using structures that make it easy to understand and parse. Second, the system must produce text that is sufficiently high quality (in terms of e.g. grammar) to be understood.

With regard to the information content of the output, the *relevance* of the results is not in the purview of the Reporter, but of the Investigator that determines the relative importance values of the various data points. At the same time, the ordering of the information within the reports is explicitly the domain of the Reporter. Following iterative changes to remove extreme cases of excessive information being included in the reports produced by intermediate versions of the Reporter, we are not aware of any major problems in selecting or structuring the contents of the reports. Wherein the feedback has pointed out any specific problems (most noticeably with long list-like paragraphs), we have modified the relevant templates and realizers to improve the quality of the output to address the comments. In general, we believe that the structuring of the reports is of at least acceptable quality and balances well the quality of the output with the other requirements imposed on the system.

To evaluate the linguistic quality of the output, feedback was sourced from the the internal domain expert testers using intermediate versions of the system. In general, the experts testers found the language of the system stilted but understandable, indicating that the linguistic quality of the system output is at least acceptable. These sentiments were We will continue to elicit and address feedback on this factor to the end of the project.

An as of yet unsolved problem is tailoring the system output to individual users' knowledge of the tools being employed. Some users would benefit from longer descriptions explaining in detail what terms such as 'Jensen–Shannon divergence' mean, and how the results should be interpreted, while including that information would be completely superfluous to others. In the present version of the Reporter, we have attempted to overcome the tailoring problem by including tooltips that describe several terms when the terms are moused over in the text. The tooltips give brief definitions and some instructions on how to interpret values, such as what the range of a metric is and how the extrema are to be interpreted. While this does not completely remove the problem (as even the tooltip contents might benefit from tailoring), we believe it alleviates it.

Based on the above consideration, we believe the Reporter shows good fitness for its intended purpose. At the same time, the feedback received indicates that further improvements could be made to the system.

In the broader natural language generation literature, NLG systems are most commonly evaluated using automated metrics that compare the output of the NLG system to some known human-produced gold standard text for a known system input. Such comparisons are conducted using any of a plethora of evaluation metrics that attempt to address the inherent difficulty of determining the similarity of two human texts: small word-level choices (e.g. introducing a negation, changing a verb for an antonym, etc.) can have immense changes for the meaning of the text, while simultaneously sentences can often be rephrased almost completely (changing the bulk of the individual words) without affecting the overall meaning. While a plethora of metrics have been proposed to complete this difficult task, it has been observed that these metrics should be used with caution and their limitations acknowledged [21]. As the Reporter is a novel system addressing a novel research need, we lack a gold-standard corpus of system outputs that would be needed to conduct automated metric-based evaluation of the Reporter. Simultaneously creating such a corpus would be prohibitively expensive and resource intensive, as the creation effort would need to be conducted by domain experts who would need to, in the worst case, comb through the thousands of data points provided to the Reporter as its input to produce even a single gold-standard output. For every input, a large amount of such gold-standard texts would be needed, as a fundamental assumption of the automated evaluation metrics is that the gold standard texts cover the whole space of acceptable outputs. As such, we interpret that automated evaluation of the Reporter is fundamentally untenable at this point. However, as human evaluations are commonly viewed as more reliable than automated evaluations in NLG literature [42], we believe this does not reduce the credibility of our analysis above.

10. Conclusions

In this Deliverable we have described the finalized version of the NewsEye Personal Research Assistant's Reporter component, and how it relates to the rest of the NewsEye project and especially the other components of the Personal Research Assistant. The Reporter follows a modularized pipeline architecture for data-to-text NLG, which allows for guarantees regarding output correctness, extensibility, explainability and multilinguality.

We believe the Reporter's architecture is sound and presents a highly suitable report generation approach given the state of the art in NLG, the requirements of the NewsEye context, and the limitations imposed by lack of suitable training data for any machine learning based NLG methods. While the output of the Reporter is relatively stiff and formulaic, it strikes a balance between being able to adapt to

the various experiments conducted by the Investigator and any changes in the available analytical tools while ensuring that the produced texts are truthful to the underlying data. While some other methods of NLG might have resulted in more fluent textual output, we believe that the risk of misleading that would have risen from the use of those methods would have been unacceptable given the context in which the Reporter is used.

Overall, we believe that the Reporter component is a successful application of NLG techniques to an extremely challenging domain and application context. This is not to say that future improvements are not possible. However, the extensible and modular nature of the Reporter makes any such changes relatively easy to implement as they are identified. Finally, the modular nature allows future efforts to integrate machine learning based NLG methods into the Reporter pipeline as those technologies mature.

References

[1] Leo Leppänen and Hannu Toivonen. “A Baseline Document Planning Method for Automated Journalism”. In: *Proceedings of the 23rd Nordic Conference on Computational Linguistics (NoDaLiDa)*. Linköping University Electronic Press. 2021.

[2] Ross Turner, Somayajulu Sripada, Ehud Reiter, and Ian P Davy. “Using spatial reference frames to generate grounded textual summaries of georeferenced data”. In: *Proceedings of the fifth international natural language generation conference*. Association for Computational Linguistics. 2008, pp. 16–24.

[3] François Portet, Ehud Reiter, Albert Gatt, Jim Hunter, Somayajulu Sripada, Yvonne Freer, and Cindy Sykes. “Automatic generation of textual summaries from neonatal intensive care data”. In: *Artificial Intelligence* 173.7-8 (2009), pp. 789–816.

[4] Catalina Hallett and Donia Scott. “Structural variation in generated health reports”. In: *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*. Association for Computational Linguistics. 2005, pp. 33–40. URL: http://aclweb.org/anthology/I05-5005.

[5] Ehud Reiter, Somayajulu Sripada, Jim Hunter, Jin Yu, and Ian Davy. “Choosing words in computer-generated weather forecasts”. In: *Artificial Intelligence* 167.1-2 (2005), pp. 137–169.

[6] Jose Coch. “Multimeteo: multilingual production of weather forecasts”. In: *ELRA Newsletter* 3.2 (1998).

[7] Eli Goldberg, Norbert Driedger, and Richard I Kittredge. “Using natural-language processing to produce weather forecasts”. In: *IEEE Expert* 9.2 (1994), pp. 45–53.

[8] Lidija Iordanskaja, Myunghee Kim, Richard Kittredge, Benoit Lavoie, and Alain Polguere. “Generation of extended bilingual statistical reports”. In: *Proceedings of the 14th conference on Computational linguistics-Volume 3*. Association for Computational Linguistics. 1992, pp. 1019–1023.

[9] Dietmar Rösner. *The automated news agency: SEMTEX–a text generator for German*. Martinus Nijhoff Publishers, 1987.

[10] Henry H. Eckerson. *The Ultimate Guide to Natural Language Generation Definitions, Trends, and Products*. 2017.

[11] Stefanie Sirén-Heikel, Leo Leppänen, Carl-Gustav Lindén, and Asta Bäck. “Unboxing news automation: Exploring imagined affordances of automation in news journalism”. In: *Nordic Journal of Media Studies* 1.1 (2019), pp. 47–66.

[12] Ehud Reiter and Robert Dale. *Building natural language generation systems*. Studies in Natural Language Processing. Cambridge University Press. 2000.

[13] Ehud Reiter. “An architecture for data-to-text systems”. In: *Proceedings of the Eleventh European Workshop on Natural Language Generation*. Association for Computational Linguistics. 2007, pp. 97–104.

[14] Albert Gatt and Emiel Krahmer. “Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation”. In: *Journal of Artificial Intelligence Research* 61 (2018), pp. 65–170.

[15] Dimitra Gkatzia. “Content Selection in Data-to-Text Systems: A Survey”. In: *arXiv preprint* (2016). Available at https://arxiv.org/abs/1610.08375.

[16] Anja Belz and Eric Kow. “Extracting parallel fragments from comparable corpora for data-to-text generation”. In: *Proceedings of the 6th International Natural Language Generation Conference*. Association for Computational Linguistics. 2010, pp. 167–171.

[17] Nina Dethlefs. “Context-Sensitive Natural Language Generation: From Knowledge-Driven to Data-Driven Techniques”. In: *Language and Linguistics Compass* 8.3 (2014), pp. 99–115.

[18] Ondřej Dušek, Jekaterina Novikova, and Verena Rieser. “Findings of the E2E NLG challenge”. In: *arXiv preprint arXiv:1810.01170* (2018).

[19] Ehud Reiter. *Hallucination in Neural NLG*. https://ehudreiter.com/2018/11/12/hallucination-in-neural-nlg/. Accessed: 2020-03-02. 2018.

[20] Chia-Wei Liu, Ryan Lowe, Iulian V Serban, Michael Noseworthy, Laurent Charlin, and Joelle Pineau. “How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation”. In: *arXiv preprint arXiv:1603.08023* (2016).

[21] Ehud Reiter and Anja Belz. “An investigation into the validity of some metrics for automatically evaluating natural language generation systems”. In: *Computational Linguistics* 35.4 (2009), pp. 529–558.

[22] Feng Nie, Jin-Ge Yao, Jinpeng Wang, Rong Pan, and Chin-Yew Lin. “A simple recipe towards reducing hallucination in neural surface realisation”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019, pp. 2673–2679.

[23] Ondřej Dušek, David M Howcroft, and Verena Rieser. “Semantic Noise Matters for Neural Natural Language Generation”. In: *Proceedings of the 12th International Conference on Natural Language Generation*. 2019, pp. 421–426.

[24] Ratish Puduppully, Li Dong, and Mirella Lapata. “Data-to-text generation with content selection and planning”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 6908–6915.

[25] Ratish Puduppully and Mirella Lapata. “Data-to-text Generation with Macro Planning”. In: *arXiv preprint arXiv:2102.02723* (2021).

[26] Thiago Castro Ferreira, Chris van der Lee, Emiel van Miltenburg, and Emiel Krahmer. “Neural data-to-text generation: A comparison between pipeline and end-to-end architectures”. In: *arXiv preprint arXiv:1908.09022* (2019).

[27] Leo Leppänen, Myriam Munezero, Mark Granroth-Wilding, and Hannu Toivonen. “Data-Driven News Generation for Automated Journalism”. In: *Proceedings of the 10th International Conference on Natural Language Generation*. 2017, pp. 188–197.

[28] IETF RFC 8259 / STD 90. *The JavaScript Object Notation (JSON) Data Interchange Format*. Standard. Internet Engineering Task Force, Dec. 2017.

[29] ISO/IEC 21778:2017(E). *Information technology – The JSON data interchange syntax*. Standard. The International Organization for Standardization & the International Electrotechnical Commission, Nov. 2017.

[30] William C Mann and Sandra A Thompson. “Rhetorical structure theory: Toward a functional theory of text organization”. In: *Text-Interdisciplinary Journal for the Study of Discourse* 8.3 (1988), pp. 243–281.

[31] Peter White. “Narrative impulse in mass-media ‘hard news’ reporting”. In: *Genre and institutions: Social processes in the workplace and school* (2005), pp. 101–123.

[32] Mika Hämäläinen. “UralicNLP: An NLP Library for Uralic Languages”. In: *Journal of Open Source Software* 4.37 (2019), p. 1345. DOI: 10.21105/joss.01345.

[33] Elvys Linhares Pontes. “Compressive Cross-Language Text Summarization”. Theses. Université d’Avignon, Nov. 2018. URL: https://hal.archives-ouvertes.fr/tel-02003886.

[34] Yang Liu and Mirella Lapata. “Hierarchical Transformers for Multi-Document Summarization”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 5070–5081. DOI: 10.18653/v1/P19-1500. URL: https://www.aclweb.org/anthology/P19-1500.

[35] Rada Mihalcea and Paul Tarau. “TextRank: Bringing Order into Texts”. In: *Proceedings of EMNLP 2004*. Ed. by Dekang Lin and Dekai Wu. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 404–411. URL: http://www.aclweb.org/anthology/W04-3252.

[36] Jaime Carbonell and Jade Goldstein. “The Use of MMR, Diversity-based Reranking for Reordering Documents and Producing Summaries”. In: *SIGIR*. 1998, pp. 335–336.

[37] Derek Miller. *Leveraging BERT for Extractive Text Summarization on Lectures*. 2019. arXiv: 1906.04165 [cs.CL].

[38] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. “Transformers: State-of-the-Art Natural Language Processing”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[39] George Giannakopoulos, Mahmoud El-Haj, Benoit Favre, Marina Litvak, Josef Steinberger, and Vasudeva Varma. “TAC2011 MultiLing Pilot Overview”. In: *Proceedings of the Fourth Text Analysis Conference, TAC 2011, Gaithersburg, Maryland, USA, November 14-15, 2011*. 2011. URL: http://www.nist.gov/tac/publications/2011/additional.papers/Summarization2011%5C_MultiLing%5C_overview.proceedings.pdf.

[40] Chin-Yew Lin. “ROUGE: a Package for Automatic Evaluation of Summaries”. In: *Proceedings of the Workshop on Text Summarization Branches Out (WAS 2004)*. 2004, pp. 74–81.

[41] Craig Thomson and Ehud Reiter. “A Gold Standard Methodology for Evaluating Accuracy in Data-To-Text Systems”. In: *Proceedings of the 13th International Conference on Natural Language Generation*. 2020, pp. 158–168.

[42] David M Howcroft, Anja Belz, Miruna-Adriana Clinciu, Dimitra Gkatzia, Sadid A Hasan, Saad Mahamood, Simon Mille, Emiel van Miltenburg, Sashank Santhanam, and Verena Rieser. “Twenty years of confusion in human evaluation: NLG needs evaluation sheets and standardised definitions”. In: *Proceedings of the 13th International Conference on Natural Language Generation*. 2020, pp. 169–182.

A. Reporter API Description

A.1. Endpoints

- `GET` /**api**/**languages** - List supported languages
- `GET` /**api**/**formats** - List supported formats
- `POST` /**api**/**report** - Produce a report from `multipart/form-data` input
- `POST` /**api**/**report**/**json** - Produce a report from `application/json` input

A.2. `GET` /api/languages

Describes the languages supported by the Reporter. All languages in the response are valid to be used as the `language` parameter in the `POST` /**api**/**report** request.

A.2.1. Parameters

None

A.2.2. Example Response

```
{
  "languages": [
    "en",
    "fi",
    "de",
    "fr"
  ]
}
```

A.3. `GET` /api/formats

Describes the text formatting options supported by the Reporter. All formats in the response are valid to be used as the `format` parameter in the `POST` /**api**/**report** request.

A.3.1. Parameters

None

A.3.2. Example Response

```
{
  "formats": [
    "p",
    "ol",
    "ul"
  ]
}
```

A.4. POST /api/report

Produces a natural language report from analytical results. The response consists of three mandatory fields, `language` which describes the language of the report, `body` which contains a list of the body text segments of the report as HTML and `head` which contains a list of the headers assoaciated with each body text segment. In other words, the first body text segment (of potentially multiple paragraphs) is associated with the first header.

This endpoint assumes the body of the request has `Content-Type` set to `multipart/form-data`, with the contents of the `data` field being a JSON string. Each of the fields (defined below) is presented as it's own form field.

The response can also contain an additional `errors` field, which describes any errors encountered during the generation process. It is possible for there to be multiple errors, as each header and section of the document is generated separately, as noted in Section 6.

A.4.1. Parameters

Field	Description
`language`	The language the report should be written in. Valid values are those returned by the `GET` **/api/languages** endpoint.
`format`	The format of the body of the report. Valid values are those returned by `GET` **/api/formats** endpoint. Supported values are '`p`' for paragraphs of text, '`ul`' for a list of bullet points and '`ol`' for a list with numbered elements.
`data`	A JSON list of analysis results returned by the Investigator as a JSON object. Format of the individual results is dictated by Deliverable D5.3, which describes the initial version of the Investigator.
`links`	Optional field. Setting the value to 'True' produces a report containing link tags (see Section 5.9. Omission of the field, or setting it to 'False', results in a text without link tags.

A.4.2. Example Response

```
{
  "language": "en",
  "body": [
    "<p>...</p>",
    ...
  ],
  "head": [
    "<h1>...</h1>",
    ...
  ]
}
```

A.5. POST /api/report/json

Identical to `/api/report` with the exception that the request is expected to have a `Content-Type` of `application/json`. The body is then expected to consist of a single JSON object, with fields corresponding to the fields defined in A.4.1. See A.4.2 for example output.

B. A Baseline Document Planning Method for Automated Journalism

Published as Leo Leppänen and Hannu Toivonen. "A Baseline Document Planning Method for Automated Journalism". In: *Proceedings of the 23rd Nordic Conference on Computational Linguistics (NoDaLiDa)*. Linköping University Electronic Press. 2021

A Baseline Document Planning Method for Automated Journalism

Leo Leppänen
University of Helsinki
Department of Computer Science
leo.leppanen@helsinki.fi

Hannu Toivonen
University of Helsinki
Department of Computer Science
hannu.toivonen@helsinki.fi

Abstract

In this work, we present a method for content selection and document planning for automated news and report generation from structured statistical data such as that offered by the European Union's statistical agency, Eurostat. The method is driven by the data and is highly topic-independent within the statistical dataset domain. As our approach is not based on machine learning, it is suitable for introducing news automation to the wide variety of domains where no training data is available. As such, it is suitable as a low-cost (in terms of implementation effort) baseline for document structuring prior to introduction of domain-specific knowledge.

1 Introduction

Automated generation of news texts from structured data – often referred to as 'automated journalism' (Graefe, 2016; Dörr, 2015; Caswell and Dörr, 2018) or 'news automation' (Linden, 2017; Sirén-Heikel et al., 2019; Dierickx, 2019) – is of great interest to various news producers. It is seen as a way of 'providing efficiency, increasing output and aiding in reallocating resources to pursue quality journalism' (Sirén-Heikel et al., 2019, p. 47). While data-to-text NLG systems are still far from common especially among the smaller, regional news industry players, at least among the larger newsrooms the use of NLG approaches has clearly been established (Fanta, 2017).

While secrecy in the industry makes it difficult to establish the commercial reality as an outsider, the limited available evidence indicates that commercial automated journalism is mostly done using rule-based methods despite a surge of academic interest in increasingly complex neural methods for NLG (e.g. Puduppully et al., 2019; Ferreira et al., 2019): Interviews of news automation users indicate that the employed methods are mostly based on templates (Sirén-Heikel et al., 2019), as are the few open source code repositories of real-world news automation systems (Yleisradio, 2018). Indeed, some NLG industry experts believe that especially end-to-end neural models do not match customer needs at this time (Reiter, 2019).

Contributing factors include a lack of control (Reiter, 2019); issues with hallucination of non-grounded output (Nie et al., 2019; Dušek et al., 2019; Reiter, 2018); the difficulty in surgically correcting any issues identified in trained neural models beyond additional training; as well as the difficulty of establishing what the 'worst case' performance of a neural model is.

In addition, we believe that that while neural NLG methods are theoretically highly transferable, the *practical* transferability of neural NLG solutions to many news domains is limited by a lack of training data. While newsrooms have extensive archives of news text, these are rarely associated with the matching data that is the 'input' for each piece of news text (E.g., MacKová and Sido, 2020, pp. 43–44, Kanerva et al., 2019, p. 247). At the same time, the non-trainable methods for NLG, too, suffer from difficulties in transferability and reusability (Linden, 2017).

In this work, we investigate document planning (selecting what content and in what order should appear in the document) for structured, statistical data-to-text NLG in the context of automated journalism targeting human journalists. We are not in search of a perfect method, but rather something that is relatively easy to implement as a subdomain-independent baseline and which can then be enhanced with domain-specific processing later-on. Such a method would make it easier to introduce automated journalism solutions to completely new subdomains within the larger statistical data domain.

2 Structuring Hard News

When queried for insight into news structure, journalists and academics often recite the concept of the "(inverted) news pyramid", where the news article is structured so that the order in which information appears in the text reflects the journalist's belief about the importance of the piece of information (Thomson et al., 2008). While the precise origin of the structure is not clear (Pöttker, 2003), it has become so prototypical that it is held self-evident in the journalistic trade literature: "*Every journalist knows how to write a traditional news text: start with the most important thing and continue until you have either said everything relevant or the space reserved for the story runs out*" (Sulopuisto, 2018, translated from Finnish).

A more rigorous analysis of the structures employed in 'hard' news is presented by White (1997), who argues that hard news articles have an 'orbital' structure consisting of a *nucleus* which represents the main point of the article and *satellites* that give context and additional information about the nucleus. White (1997) assigns the role of the nucleus to the combination of the headline and the lead paragraph of the article, and describes the subsequent paragraphs as the satellites. White (1997) identifies five possible relations between a satellite and the nucleus: elaboration, cause-and-effect, justification, contextualization and apprisal. Thomson et al. (2008), in turn, identify that the satellites can elaborate, reiterate, describe causes or consequences, contextualize or provide additional assessment. An important observation is that – as indicated by 'orbital' – these satellites are relatively freely reorderable without affecting readability or meaning. Together, these two observations indicate that a good document plan for hard news (1) prioritizes more newsworthy items and (2) contains some overarching theme (exemplified by the nucleus) so that the text as a whole is coherent, i.e. the satellites are in some way related to the nucleus.

The relations identified by White (1997) and Thomson et al. (2008) are highly similar to those identified in the more general Rhetorical Structure Theory (RST) (Mann and Thompson, 1988), which uses similar nucleus-satellite terminology. However, whereas White (1997) and Thomson et al. (2008) analyze news text on the level of paragraphs, RST can be applied on a more fine-grained level to much shorter text spans. As RST shows that similar relations can be applied on a sub-paragraph level, we hypothesize that a reasonably approximation of a news article might be constructed by applying White's (1997) orbital theory also *within* paragraphs, by considering the first *sentence* of the paragraph a nucleus, and the others as satellites.

Importantly, we interpret the orbital theory of news structuring to suggest that – as the satellites are freely orderable – the actual *type* of relation is not as important for document planning as knowing that *some* relation exists between the satellite and the nucleus. We hypothesize that while identifying whether a specific (RST) relation exists between two arbitrary pieces of information requires domain knowledge, an approximation of whether two arbitrary pieces of information are related in *some* way could be obtained by inspecting their similarity in a domain-independent fashion.

That is, we expect that a piece of information regarding the US health care funding in 2020 is more likely to be related in *some way* to a piece of information discussing the US health care funding in 2020 than to another piece of information discussing the health care funding in Sweden in 1978. If a heuristic or similarity measure identifying such relations could be identified, it could be used together with some estimate of newsworthiness to construct paragraph and document plans that seek to maximize both the key aspects identified above: newsworthiness and the relatedness of the content.

As noted in the introduction, there is a distinction between the theoretical and the practical transferability of neural processing methods. We believe that a good baseline document planning and content selection approach should avoid the need for training data present in the many of recently proposed document planning and content selection approaches. This rules out as unsuitable most recent work that are based on learning from an aligned corpus of data and human-written texts, such as Angeli et al. (2010), Konstas and Lapata (2013), Wiseman et al. (2017), Zhang et al. (2017), Li and Wan (2018), Dou et al. (2018) and Puduppully et al. (2019).

Outside of these trainable approaches, to our knowledge, most other document planning approaches are based on '*hand-engineered*' (Konstas and Lapata, 2013), domain-specific methods. A highly relevant survey of various document planning methods is presented by Gkatzia (2016). While these previous works are – to at least some degree – domain-specific, they establish concepts

and ideas that are highly relevant for our goal. Both Hallett et al. (2006) and Gatt et al. (2009) describe a core set of information, called 'summary spine' or 'key events', that they hold as more important than the rest of the available information. They, as well as Banaee et al. (2013), also employ a numeric estimate of importance. Demir et al. (2010) identify that content already selected for inclusion in the document plan affects how well suited so-far unselected content is for inclusion. Sripada et al. (2003) identify Gricean maxims (Grice, 1975) as providing requirements for document planning and content selection.

3 Context

Our work on document planning is done in the context of a series of data-to-text NLG applications producing short highlights of structured statistical data. Importantly, the applications are intended to be deployed in contexts where they must be able to produce texts highlighting between 10 and 30 data points from datasets measured in 100.000s of data points. The resulting texts are intended to both alert journalists to potential news and to provide them with a starting place from which to write the final news text.

Our system, adapted from Leppänen et al. (2017a), is based on a pipeline of components with dedicated responsibilities similar to those described by Reiter and Dale (2000) and Reiter (2007). For this work, the relevant part of the architecture is the Document Planner component. This component receives as input two sets of *message* data structures, an example of which is shown in Table 1.[1] The messages are extracted automatically from tables of statistical data obtained from Eurostat.

The *core set* contains messages that are known to be highly relevant to the generation task. Unlike the "summary spine' of Hallett et al. (2006), the set is unlinked and unordered, and not all members of the set are guaranteed to be included in the document plan. The *expanded set*, contains messages that *can* be, but are not guaranteed to be, relevant for the document. Expressed using the terminology from Section 2, we assume that only messages in the core set can be nuclei, while messages from either set can be satellites.

These core and expanded sets are determined automatically from user input. When requesting a new text, the user of the system must define a dataset the text is to be generated from, for example the consumer price data available from Eurostat. This dataset is then divided into the core set and the expanded set by the user when they select what country the generated text should focus on. For example, if the user were to select that the text should discuss French consumer prices, the core set would contain all data from the consumer price dataset that pertains directly to France, while the rest of the consumer price dataset (including data pertaining to the UK, Finland, Croatia, etc.) would be set as the expanded set.

We estimate each message's 'newsworthiness' using the Interquartile Range based method described by Leppänen et al. (2017b) with the values scaled to have mean 0 and standard deviation 1 for the purposes of this computation. The resulting value is conceptually similar to 'importance' of Gatt et al. (2009) and 'risk' of Banaee et al. (2013). The IQR based method compares each data point in turn to a larger distribution, giving it higher scores the further it is from the area between the first and the third quartile of the larger distribution. Values between the quartiles are given a minimal, uniform, score that is dependent on the shape of the distribution. In other words, higher IQR values indicate that the value is more of an outlier compared to the rest of related data in the dataset. As such, it captures a degree of 'unexpectedness', which is an important aspect of newsworthiness (Galtung and Ruge, 1965).

We do not use the domain-specific parts of the method described by Leppänen et al. (2017b). That is, we make no value judgement of whether messages pertaining to French consumer prices are more newsworthy than messages pertaining to Croatian consumer prices, nor do we make judgements of whether changes in the price of education are more or less newsworthy than changes in the price of alcohol and tobacco. However, we do weight the scores so that messages with the `timestamp` field being closer to present receive higher weights, as recency is an important aspect of newsworthiness. While we have described our method for computing the `newsworthiness` value in some detail, we emphasize that for the rest of this article we only assume that the `newsworthiness` values are non-negative and that higher values indicate higher newsworthiness.

More crucially for the method described be-

[1] The concrete implementation details are somewhat more complex. We omit details irrelevant for this work.

low, we specify that the `value_type` fields (which describe how the messages' values are to be interpreted) contain members of a hierarchical taxonomy of data types represented as colon-separated hierarchies of labels. For example, the `value_type` field value `health:cost:hc2:mio_eur` would indicate that the number in the `value` field is the amount of money (`cost`), measured in millions of euros (`mio_eur`), spent by some nation (as defined by the `location` and `location_type` fields) on rehabilitative care (`hc2`) in some time period (as defined by the `timestamp` and `timestamp_type` fields) and that this is part of the larger health care topic (`health`). In our case, these labels are automatically established from the headers of the input data tables.

The goal of document structuring is to produce a three-level tree-structure with ordered children. The root node corresponds to the document as a whole and the mid-level structures correspond to paragraphs. The leaves are the messages selected for inclusion in the document. While the messages have not yet, at this stage, been associated with any linguistic structures, they can be conceptualized as being phrases or very short sentences. We are thus concurrently determining both the content and the structure the document.

We emphasize that our applications are employed in domains where they must be able to select some 10-30 messages from a pool of potential messages numbering in 100,000s. Given infinite computational resources, it would be preferential to construct all possible document plans and then score them in some fashion. This, however, is infeasible given the size of the search space. Previously, other authors have employed, for example, stochastic searches with significantly smaller search spaces (Mellish et al., 1998). Indeed, some kind of a beam search approach could be very useful in smartly searching a subset of the search space. However, we have thus far been unable to identify a document-level metric that adequately balances the 'total amount of newsworthiness' in a text with the length of the text, a requirement for beam search.

4 Research Objective

Based on the above considerations, our main goal is to identify a widely applicable method for content selection and document planning that matches the following requirements:

REQ1: The method needs to be highly performant

REQ2: The method should not be dependent on domain knowledge

REQ3: The document should have a theme

REQ4: The document should have multiple paragraphs but not be excessively long

REQ5: The paragraphs should have distinct themes related to the document theme

REQ6: The paragraph themes should be newsworthy in their own right

REQ7: The paragraphs should not be excessively long or short

REQ8: All messages should relate to the paragraph theme

REQ9: All messages should be newsworthy

REQ10: Within each paragraph, the messages should be presented in an order that produces a coherent narrative

Again, we emphasize that our goal is not to identify a method that is optimal for any specific scenario, but rather to determine a baseline method that is *adequate* for a broad spectrum of applications and sub-domains.

5 A Baseline Approach to Document Planning

Optimally, we would wish to produce some sort of a *globally optimal* document plan. However, as discussed above, this would entail significant computational costs and require a scoring function applicable to the document as a whole. As such, we propose a method for producing document plans in a greedy, linear, and iterative fashion. At every stage, decisions are made considering only a limited local context, thus avoiding the need for a method of determining the global quality of the document plan, thus fulfilling REQ1 ('The method needs to be highly performant').

The document's overall theme, in our use case, is selected by the user who initiates the generation task. In initiating the task, the users selects both a dataset and a focus location. The generation process then derives the *core messages* and *expanded messages* sets (the inputs to the Document Planner, see Section 3) so that both sets discuss the dataset

Field	Description	Example value
`where`	What location the fact relates to	Finland
`where_type`	What the type of the location is	country
`timestamp`	The time (or time range) the fact relates to	2020M05
`timestamp_type`	The type of the timestamp	month
`value`	A (usually) numeric value	0.01
`value_type`	Interpretation of `value`	cphi:hicp2015:cp-hi02:rt01
`newsworthiness`	An estimate of how newsworthy the message is	1

Table 1: An example of a message. The hypothetical message states that in the fifth month of 2020, in Finland, the consumer price index, using the year 2015 as the start of the index, of alcoholic beverages and tobacco changed by 0.01 points with respect to the value of the index during the previous month.

indicated by the user (i.e. messages from other datasets are not generated) and that the core set contains messages pertaining to the user's indicated focus location, while messages pertaining to all other locations are in the expanded set. This fulfills REQ3 ('The document should have a theme'). This step is also independent of the specific subdomain, thus fulfilling REQ2 ('The method should not be dependent on domain knowledge'). This step thus fulfills all the relevant requirements. Next, we'll describe how both the first and subsequent paragraphs can be planned in a way consistent with the requirements defined above.

5.1 Planning the First Paragraph

At the start of the document planning process, we select the most newsworthy message from the *core messages* set to act as the nucleus (n_1) of the first paragraph (p_1). This nucleus establishes the theme of the first paragraph as follows: We inspect the `value_type` field of this first nucleus n_1, and retrieve a prefix `Prefix`(n_1). The prefix is the least amount of colon-separated labels wherein the total amount of prefixes in the core set is greater than the minimal amount of paragraphs a document can have, in our case two. In our case, as a consequence of our label hierarchy, this is always the first three colon-separated units. For the message shown in Table 1, the prefix would thus be `cphi:hicp2015:cp-hi02`, meaning that the first paragraph's theme would be the prices of alcoholic beverages and tobacco. This fulfills REQ5, 'the paragraphs should have distinct themes related to the document theme' for the first paragraph.

Next, the first paragraph is completed with satellites from the union of the *core messages* and the *expanded messages* sets. These satellites are initially filtered so that only messages that have the same prefix as the nucleus n_i are considered in paragraph p_i to fulfill REQ8 ('All messages should relate to the paragraph theme'). The satellites are then selected in a linear, greedy, and iterative manner to fulfill REQ1.

For selecting the k'th satellite to a partially constructed paragraph already containing $k-1$ satellites and one nucleus, we consider both the newsworthiness of the available messages (REQ9), as well as how well they would fit the already constructed segment (REQ8). Observing only the newsworthiness would produce a highly incoherent narrative, whereas focusing only on the narrative risks leaving out highly important information.

Following the reasoning in Section 2, we assume that two subsequent messages are more likely to form a good narrative if they are similar. As such, we need a method for weighing the message's newsworthiness by the similarity of the message to the last message of the under-construction paragraph, thus balancing the requirements of REQ8 and REQ9. In terms of the message objects described in Table 1, it seems to us that the intuitive aspects of similarity are related to the degree of similarity within the 'meta' fields such as `timestamp`, `location` and `value_type`.

For the `timestamp` and `location` fields, we can state that two messages that have identical values in the fields are more similar that two messages that are otherwise the same but have distinct values for said fields. We call this the *contextual* similarity of the messages, and the fields the *contextual fields* (F_c), as these fields provide us access to the larger context in which the `value` and `value_type` fields can be interpreted. Contextual similarity captures the notion that it is likely better to follow a fact about French healthcare spending in 2020 with another piece of information about France in 2020,

rather than about Austria in 1990.

In more precise terms, we propose the following weighing scheme for contextual similarity: The similarity $sim_c(A, B)$ of two messaged A and B is the product of weights $w_f > 1$ for each field f among the contextual fields F_c, where both A and B have the same value for the field:

$$sim_c(A, B) = \prod_{\{f \in F_c | A.f = B.f\}} w_f \quad (1)$$

This value strictly increases as more fields are shared between A and B. We explicitly define the similarity to be zero if there are no fields f where A and B share a value. If w_f is a uniform value for all fields f, this scheme is completely domain-agnostic. Setting different weights w_f for each field $f \in F_c$ allows for encoding some domain knowledge about which fields are the most important for the text, thus providing a method for producing more tailored texts at the cost of slightly violating REQ2. In our case study, we set $w_{timestamp} = 1.1$ and $w_{location} = 1.5$.

The above consideration of similarity still ignores valuable information available from the `value_type` field, which describes how the value in the `value` field is to be interpreted. Denoting `health:cost:hc2:mio_eur` (the cost of rehabilitative care in millions of euros) by T_1, consider its similarity to T_2 = `health:cost:hc2:eur_hab`, the cost of rehabilitative care as euros per inhabitant, and T_3 = `health:cost:hc41:mio_eur`, the cost of health care related imaging services in millions of euros. Intuitively, T_1 and T_2 are thematically closer than T_1 and T_3. We model this similarity between two facts A and B simply as

$$sim_t(A, B) = \frac{1}{s(A, B)} \quad (2)$$

where $s(A, B)$ is the length – in colon-separated units – of the unshared suffix between A and B's `value_type` fields. That is, $s(T_1, T_2) = 1$ whereas $s(T_1, T_3) = 2$. We specify that $sim_t(\cdot, \cdot)$ is zero for all pairs without any shared prefix.

Our formulation of $sim_t(\cdot, \cdot)$ was influenced by the observation that in our context the messages' `value_type` values have a constant number of colon-separated segments. In cases where the lengths of the `value_type` values differ, an alternative formulation of

$$sim'_t(A, B) = \frac{2p(A, B)}{\ell(A) + \ell(B)} \quad (3)$$

where $\ell(\cdot)$ provides the length of the `value_type` value, and $p(\cdot, \cdot)$ is the length of shared *prefix* between A and B, both measured as colon-separated units, might be preferable if also more complex.

When considering whether the k'th satellite s_i^k of paragraph p_i should be a specific candidate $c \in C$, where C is all so far unused messages, we can combine the similarity metrics with the newsworthiness of c into a general fitness value as follows:

$$\begin{aligned} fit(c, x) = c.newsworthiness \\ \times sim_c(c, x) \\ \times sim_t(c, x) \\ \times set\_penalty(c) \end{aligned}$$

The $set\_penalty(c)$ factor depends on whether the message originates from the *core messages* set, or the *extended messages* set. For messages originating from the core message set, the penalty is 1. For messages originating from the extended messages set, the penalty is $\frac{1}{dist+1}$, where $dist$ is the distance from the previous core message.

The final score describing how good of an addition c would be as the kth satellite of the ith paragraph s_i^k is then obtained by taking the average of fitnesses of c in relation to both the nucleus n_i and the previous satellite s_i^{k-1} by computing:

$$score(c, n_i, s_i^{k-1}) = \frac{fit(c, n_i) + fit(c, s_i^{k-1})}{2}$$

This maximizes the newsworthiness of the paragraph's contents (fulfilling REQ9, 'all messages should be newsworthy'), while also enforcing relatedness to the theme of the paragraph (fulfilling REQ8, 'all messages should relate to the paragraph theme') by measuring against the nucleus and with the inclusion of the *set_penalty*. By continuously measuring against the previously selected satellite, the procedure also allows for interludes to e.g. discuss highly newsworthy information related to but not strictly about the paragraph's main topic, or 'thematic drift'. It thus fulfills REQ10 ('Within each paragraph, the messages should be presented in an order that produces a coherent narrative') while also paying attention to the pyramid model of news (See Section 2).

Using $score$, the highest scoring candidate $c_{top} = \arg\max_{c \in C} score(c, n_i, s_i^{k-1})$ is then compared to both an absolute threshold t_{abs} and the newsworthiness of the nucleus n_i multiplied by relative threshold value t_{rel}. Provided that the

maximal paragraph length has not been reached, the top candidate message c_{top} is appended to the paragraph p_i as the k'th satellite s_i^k in the document plan provided that either $score(c_{top}, n_i, s_i^{k-1}) \geq t_{abs}$ or $score(c_{top}, n_i, s_i^{k-1}) \geq t_{rel} \times n_i.newsworthiness$.

These thresholds ensure that the paragraph does not stray into minutiae, whether considered in absolute terms or in relation to the nucleus of the paragraph. In cases where the minimum paragraph length has not been reached, the thresholds are ignored and the top candidate is always appended. This accounts for REQ7 ('The paragraphs should not be excessively long or short').

The above considerations take into account several free parameters, namely the maximal and minimal paragraph lengths as well as the threshold values t_{rel} and t_{abs}. In our case study, we selected the minimal and maximal paragraph lengths as 2 and 5 messages empirically by trialing out various values and observing the resulting texts. These should, naturally, be based on the genre of text and the target audience. For the threshold values we selected 0.2 and 0.5, respectively, using the same method as with the paragraph lengths above. Both the thresholds and the minimal and maximal paragraph lengths should be viewed as (manually) tuneable hyperparameters.

5.2 Planning Subsequent Paragraphs

We then proceed to generate further paragraphs in a manner highly similar to that used when planning the first paragraph. The only distinction is that, when selecting the nucleus n_i for a subsequent paragraph p_i, we obtain the message from the *core messages* set with a highest newsworthiness value that has a prefix (theme) not yet discussed among the previously planned paragraphs p_1 - p_{i-1}:

$$n_i = \arg\max_{c \in C} c.newsworthiness \tag{4}$$

where

$$C = \Big\{ c \in CoreMessages | \texttt{Prefix}(c) \notin \{\texttt{Prefix}(n_k) | k \in [1..i-1]\} \Big\} \tag{5}$$

This ensures that the different paragraphs are highly newsworthy, thus fulfilling REQ6, while also fulfilling REQ5 for having distinct themes for the different paragraphs.

As when constructing the subsequent paragraphs, the total length of the document also needs to be considered. To fulfill REQ4 ('The document should have multiple paragraphs but not be excessively long'), we employ a variation of the method described in the previous section for ending individual paragraphs. A maximal length (in our case, 3 paragraphs) ensures that the document is not allowed to grow beyond reason, whereas a minimal length (for us, 2 paragraphs) ensures that the document is not unreasonably short. After the minimal length has been reached (but not yet the maximal length), a new paragraph is only started if the nucleus of the potential paragraph has a newsworthiness value that is at least 30 % of the newsworthiness value of the first nucleus of the document. This, as with the satellites, ensures that the the document does not stray into minutiae, balancing REQs 4 and 6. the maximal and minimal lengths, as well as the 30 % threshold, were determined by manual fine-tuning and should be viewed as tuneable hyperparameters.

6 Evaluation

The method described above was implemented in a larger NLG application producing news alerts for journalists from datasets provided by Eurostat. A variation of the same application was also developed with a simplified document planner. In this simplified planner, the planner always selects the maximally newsworthy available message as the message without any early stopping threshold. Nuclei are selected from the core messages set, while satellites can be from either set. Contrasting our proposed method with this simplified method enables us to evaluate the importance of narrative coherence in the generated texts. The larger application is multilingual, but the evaluation was conducted using English language texts.

Three experts were recruited from the Finnish News Agency STT, a national European news agency, to evaluate documents on the consumer price indices in five different European nations. For all nations, the judges were shown variants produced by both our proposed method and the simplified method. One of the selected countries is the country the news agency is based in, with the assumption that the judges would have high amounts of world knowledge they would be able to use in evaluating these texts. Another variant pair describes a country that is both relatively small and geographically remote (but still within EU), with the assumption that the journalists are unlikely to

Consumer Prices in Estonia

In June 2020, in Estonia, the monthly growth rate of the harmonized consumer price index for the category 'education' was 30.8 points. It was 30.7 percentage points more than the EU average. In July 2020, it was 0.4 percentage points less than the EU average. It was -0.4 points. In May 2020, the yearly growth rate of the harmonized consumer price index for the category 'education' was -20.5 points. It was 21.9 percentage points less than the EU average.

In August 2020, the monthly growth rate of the harmonized consumer price index for the category 'housing, water, electricity, gas and other fuels' was 2.5 points. It was 2.3 percentage points more than the EU average. In North Macedonia, it was 3 percentage points more than the EU average. It was 3.2 points. Estonia had the 3rd highest monthly growth rate of the harmonized consumer price index for the category 'housing, water, electricity, gas and other fuels' across the observed countries. In Sweden, the monthly growth rate of the harmonized consumer price index for the category 'housing, water, electricity, gas and other fuels' was 3.1 points.

Figure 1: Example output regarding Eurostat statistics on consumer prices. The text contains 12 messages, selected from among 207,210 messages available during generation.

have much world knowledge about this country's consumer prices. The three other countries were selected from among those bordering the first country, with the assumption that the journalists would have some, but not much, world knowledge relating to these countries. The final output texts were not inspected prior to selecting the countries.

All of the texts used in the evaluation were generated from a copy of the same underlying Eurostat dataset, entitled 'Harmonised index of consumer prices - monthly data [ei_cphi_m]'[2] downloaded in September 2020. It contains country-level data regarding the harmonized consumer prices indices, and their change over time, for various EU nations starting from January 1996. We preprocess the data by adding monthly rankings (i.e. determine what country had the greatest, the second greatest, etc. value for a specific index category during any specific month) and comparisons to the EU average values.

As the evaluation was focused on document planning and content selection, the larger system was simplified in some respects, e.g., to not conduct complex aggregation. This was done to minimize the effect of later stages of the generation process on the evaluation. As a result, the language in the evaluated documents was relatively stilted, as exemplified by Figure 1. The only manual alteration was the addition of headings to indicate the texts' intended themes.

The judges did not receive any direct compensation but their employer, the news agency, is a member of the EU-wide EMBEDDIA research project within which parts of this work was conducted. The evaluations were conducted online. The judges were first provided with some basic information on the type of documents they were to read (i.e. that the texts are intended to be news alerts for journalists, rather than publication ready news texts), the length of the task, etc. All instructions were in the judges' native language, in this case Finnish. The judges were not told which texts were produced by which variants nor how many variants were being tested. Following this, the judges were shown the documents one by one. For each document, the judges were asked to indicate their agreement with the following statements (translated from Finnish):

Q1: The text matches the heading

Q2: The text is coherent

Q3: The text lacks some pertinent information

Q4: The text contains unnecessary information

Q5: The text has a suitable length

For Q1–Q4, the judges indicated their agreement on a 7-point Likert scale ranging from 1 ('completely disagree') to 7 ('completely agree'). For Q5, the answers were provided on 5-point scale ranging from 1 ('clearly too short') to 3 ('length is suitable') to 5 ('clearly too long'). In addition, the judges were able to provide textual feedback for each individual text, as well as for the evaluation task as a whole. The judges' answers to Q1 – Q5, are aggregated in Table 2.

The results indicate that the proposed method statistically significantly increases the document's coherence (Q2, mean 4.33 vs. 1.60, median 5 vs 2), the matching of the document's content to the document's theme (Q1, mean 4.40 vs. 1.80, median 5 vs 2), and produces documents of more suitable length (Q5, mean 2.93 vs. 4.07, median 3 vs 4, with 3 being best). The proposed method also seems

[2] Available for download and browsing from `http://appsso.eurostat.ec.europa.eu/nui/show.do?dataset=ei_cphi_m`

	Our method			Baseline			
Statement	Median	Mean	SD.	Median	Mean	SD.	p_{MWU}
Q1 (1–7, ↑)	5	4.40	1.64	2	1.80	0.41	< 0.001*
Q2 (1–7, ↑)	5	4.33	1.76	2	1.60	0.51	< 0.001*
Q3 (1–7, ↓)	4	4.47	1.81	6	5.80	1.42	0.049
Q4 (1–7, ↓)	5	5.13	1.55	6	6.33	0.62	0.024
Q5 (1–5, 3 best)	3	2.93	0.59	4	4.07	0.70	< 0.001*

Table 2: Results obtained during the evaluation. Parentheses indicate answer ranges and whether the higher (↑), lower (↓) or middle values are to be interpreted as the best. The p_{MWU} column contains the (uncorrected) p-value of a two-sided Mann-Whitney U test. An asterisk indicates the p-value is statistically significant also after applying a Bonferroni correction to account for multiple tests.

to result in less unnecessary information being included in the document (Q4, mean 5.13 vs 6.33, median 5 vs 6), and in the text missing less necessary information (Q3, mean 4.47 vs 5.80, median 4 vs 6), but these effects are not statistically significant after correcting for multiple comparisons with the Bonferroni correction. We hypothesize this difference would become significant in a larger-scale evaluation.

The free-form textual feedback provided by the judges, as expected, indicates that the texts could be further improved. For example, in the case of the text shown in Figure 1, the judges called for a sentence explicitly noting that North Macedonia had the highest monthly growth rate. In addition, they noted it might be better to produce distinct, even shorter, texts as 'news alerts' while reserving the evaluated texts for use as a starting point when the journalist starts writing.

7 Conclusions

In this work, we have identified a need for, and proposed, a widely applicable baseline document planning method for generating journalistic texts from statistical datasets. Our method is based on observations on the similarities between the orbital theory of news structure (White, 1997) and Rhetorical Structure Theory (Mann and Thompson, 1988). While our proposed method is likely to fall short of the performance of subdomain-specific planning methods, results indicate that it achieves adequate performance while fulfilling a set of requirements identified based on the larger application domain of news generation.

Acknowledgements

This work is supported by the European Union's Horizon 2020 research and innovation program under grant agreement No 825153, project EMBEDDIA (Cross-Lingual Embeddings for Less-Represented Languages in European News Media), and grant agreement No 770299, project NewsEye (A Digital Investigator for Historical Newspapers).

References

Gabor Angeli, Percy Liang, and Dan Klein. 2010. A simple domain-independent probabilistic approach to generation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 502–512.

Hadi Banaee, Mobyen Uddin Ahmed, and Amy Loutfi. 2013. Towards NLG for physiological data monitoring with body area networks. In *14th European Workshop on Natural Language Generation, Sofia, Bulgaria, August 8-9, 2013*, pages 193–197.

David Caswell and Konstantin Dörr. 2018. Automated journalism 2.0: Event-driven narratives: From simple descriptions to real stories. *Journalism practice*, 12(4):477–496.

Seniz Demir, Sandra Carberry, and Kathleen F. McCoy. 2010. A discourse-aware graph-based content-selection framework. In *Proceedings of the 6th International Natural Language Generation Conference*.

Laurence Dierickx. 2019. Why news automation fails. In *Computation+ Journalism Symposium, Miami, FL*.

Konstantin Nicholas Dörr. 2015. Mapping the field of algorithmic journalism. *Digital journalism*.

Longxu Dou, Guanghui Qin, Jinpeng Wang, Jin-Ge Yao, and Chin-Yew Lin. 2018. Data2text studio: Automated text generation from structured data. In

Proc. 2018 Conference on Empirical Methods in Natural Language Processing.

Ondřej Dušek, David M Howcroft, and Verena Rieser. 2019. Semantic noise matters for neural natural language generation. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 421–426.

Alexander Fanta. 2017. Putting Europe's robots on the map: automated journalism in news agencies. *Reuters Institute Fellowship Paper*, pages 2017–09.

Thiago Castro Ferreira, Chris van der Lee, Emiel van Miltenburg, and Emiel Krahmer. 2019. Neural data-to-text generation: A comparison between pipeline and end-to-end architectures. *arXiv preprint arXiv:1908.09022*.

Johan Galtung and Mari Holmboe Ruge. 1965. The structure of foreign news: The presentation of the Congo, Cuba and Cyprus crises in four Norwegian newspapers. *Journal of peace research*, 2(1):64–90.

Albert Gatt, Francois Portet, Ehud Reiter, Jim Hunter, Saad Mahamood, Wendy Moncur, and Somayajulu Sripada. 2009. From data to text in the neonatal intensive care unit: Using NLG technology for decision support and information management. *Ai Communications*, 22(3):153–186.

Dimitra Gkatzia. 2016. Content selection in data-to-text systems: A survey. *arXiv preprint*. Available at `https://arxiv.org/abs/1610.08375`.

Andreas Graefe. 2016. Guide to automated journalism.

Herbert P Grice. 1975. Logic and conversation. In *Speech acts*, pages 41–58. Brill.

Catalina Hallett, Richard Power, and Donia Scott. 2006. Summarisation and visualisation of e-health data repositories. In *UK E-Science All-Hands Meeting*.

Jenna Kanerva, Samuel Rönnqvist, Riina Kekki, Tapio Salakoski, and Filip Ginter. 2019. Template-free data-to-text generation of Finnish sports news. In *Proceedings of the 22nd Nordic Conference on Computational Linguistics*, pages 242–252, Turku, Finland. Linköping University Electronic Press.

Ioannis Konstas and Mirella Lapata. 2013. Inducing document plans for concept-to-text generation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1503–1514.

Leo Leppänen, Myriam Munezero, Mark Granroth-Wilding, and Hannu Toivonen. 2017a. Data-driven news generation for automated journalism. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 188–197.

Leo Leppänen, Myriam Munezero, Stefanie Sirén-Heikel, Mark Granroth-Wilding, and Hannu Toivonen. 2017b. Finding and expressing news from structured data. In *Proceedings of the 21st International Academic Mindtrek Conference*, pages 174–183. ACM.

Liunian Li and Xiaojun Wan. 2018. Point precisely: Towards ensuring the precision of data in generated texts using delayed copy mechanism. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1044–1055, Santa Fe, New Mexico, USA. ACL.

Carl-Gustav Linden. 2017. Decades of Automation in the Newsroom: Why are there still so many jobs in journalism? *Digital Journalism*, 5(2):123–140.

Veronika MacKová and Jakub Sido. 2020. The robotic reporter in the Czech News Agency: Automated journalism and augmentation in the newsroom. *Communication Today*, 11(1):36–53.

William C Mann and Sandra A Thompson. 1988. Rhetorical structure theory: Toward a functional theory of text organization. *Text-Interdisciplinary Journal for the Study of Discourse*, 8(3):243–281.

Chris Mellish, Alistair Knott, Jon Oberlander, and Mick O'Donnell. 1998. Experiments using stochastic search for text planning. In *Natural Language Generation*.

Feng Nie, Jin-Ge Yao, Jinpeng Wang, Rong Pan, and Chin-Yew Lin. 2019. A simple recipe towards reducing hallucination in neural surface realisation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2673–2679.

Ratish Puduppully, Li Dong, and Mirella Lapata. 2019. Data-to-text generation with content selection and planning. In *Proc. 33rd AAAI Conference on Artificial Intelligence*.

Horst Pöttker. 2003. News and its communicative quality: The inverted pyramid—when and why did it appear? *Journalism Studies*, 4(4):501–511.

Ehud Reiter. 2007. An architecture for data-to-text systems. In *Proceedings of the Eleventh European Workshop on Natural Language Generation*, pages 97–104. Association for Computational Linguistics.

Ehud Reiter. 2018. Hallucination in neural NLG. `https://ehudreiter.com/2018/11/12/hallucination-in-neural-nlg/`. Accessed: 2020-03-02.

Ehud Reiter. 2019. ML is used more if it does not limit control. `https://ehudreiter.com/2019/08/15/ml-limits-control/`. Accessed: 2020-07-25.

Ehud Reiter and Robert Dale. 2000. *Building natural language generation systems*. Studies in Natural Language Processing. Cambridge University Press.

Stefanie Sirén-Heikel, Leo Leppänen, Carl-Gustav Lindén, and Asta Bäck. 2019. Unboxing news automation: Exploring imagined affordances of automation in news journalism. *Nordic Journal of Media Studies*, 1(1):47–66.

Somayajulu G Sripada, Ehud Reiter, Jim Hunter, and Jin Yu. 2003. Generating English summaries of time series data using the Gricean maxims. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 187–196.

Olli Sulopuisto. 2018. Uutisia kortti kerrallaan. *Suomen Lehdistö*. https://suomenlehdisto.fi/uutisia-kortti-kerrallaan/.

Elizabeth A Thomson, Peter RR White, and Philip Kitley. 2008. "Objectivity" and "hard news" reporting across cultures: Comparing the news report in English, French, Japanese and Indonesian journalism. *Journalism studies*, 9(2):212–228.

Peter White. 1997. Death, disruption and the moral order: the narrative impulse in mass-media 'hard news' reporting. *Genres and institutions: Social processes in the workplace and school*, 101:133.

Sam Wiseman, Stuart Shieber, and Alexander Rush. 2017. Challenges in data-to-document generation. In *Proc. 2017 Conference on Empirical Methods in Natural Language Processing*.

Yleisradio. 2018. Avoin voitto. https://github.com/Yleisradio/avoin-voitto.

Yuhao Zhang, Victor Zhong, Danqi Chen, Gabor Angeli, and Christopher D. Manning. 2017. Position-aware attention and supervised data improve slot filling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 35–45, Copenhagen, Denmark. ACL.