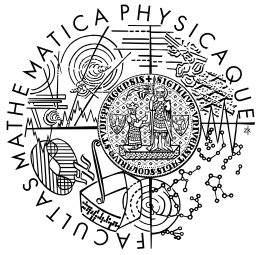


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Dan Raffl

Natural Language Generation system writing football articles

Institute of Formal and Applied Linguistics

Supervisor of the bachelor thesis: RNDr. Jiří Hana, Ph.D.
Study programme: Computer Science
Study branch: General Computer Science

Prague 2022

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

Firstly, I would like to thank my supervisor RNDr. Jiří Hana, Ph.D.for his valuable advice as well as his support. Secondly, I am grateful to every teacher I had the chance to meet at Faculty of Mathematics and Physics for their attitude and inspirational work. Last but not least, I would like to thank my family for supporting me throughout my studies.

Title: Natural Language Generation system writing football articles

Author: Dan Raffl

Institute: Institute of Formal and Applied Linguistics

Supervisor: RNDr. Jiří Hana, Ph.D., Institute of Formal and Applied Linguistics

Abstract: Journalism could become a tedious job as its main concern is to create as many articles as possible, usually prioritising quantity over quality. Some articles are quite routine and they need to exist just because most of the population is able to interpret data only in a very convenient text representation. The idea is to ease this job and generate articles, particularly about football in Czech language, automatically from non-linguistic data.

This thesis is concerned with analysing implementation of such a linguistic software and moreover offers a brief overview of a Natural Language Generation (NLG) process. The major focus of this overview is on benefits and drawbacks of different approaches to NLG as well as describing NLG tasks and its challenges you need to overcome in order to produce a similar human language (not only Czech) producing program.

Keywords: linguistics, football, NLG, natural language generation, article

Contents

Introduction

Charles Babbage, the father of the computer, had the first impulse for the invention of a mechanically calculating system at college, when he was tired of mistakes in a table of logarithms. He suggested constructing a machine powered by steam, which could process a larger number of computations than humans while avoiding making mistakes. The idea of using computers to our advantage is carried even now in a highly-paced competitive technology-driven world. Even in the field of journalism, computer science has advanced and the results are getting quite stunning. In 2020 well-known British newspaper The Guardian published an article titled “A robot wrote this entire article. Are you scared yet, human?” GPT-3, in which AI language generator GPT-3 explains why its existence is not a threat to the existence of mankind. Ignoring the main point of the article, it is well written and I doubt that anyone would recognize that humans did not write the text. Furthermore, in 2022 GPT-3’s abilities to write fluent prosaic text were described equivalent to that of a human by The New York Times Johnson and Iziev. This proves the results of AI in the field of language generating.

In this article we discuss numerous challenges of generating text in general and how to conceptually approach developing a language-producing software. This process of generating text (or some linguistic output as discussed below) is in the field of computational linguistics referred to as Natural Language Generation or NLG. These challenges are explained using various specific examples to illustrate exactly what may cause problems and how to prevent them. Also the aim of this paper is to introduce different techniques on how a NLG system can be organised and approached. Everyone who reads the paper should then be able to construct a solution for a NLG problem and especially be aware of the strengths or weaknesses of the solution.

Furthermore, one specific implementation of NLG is presented along with its analysis. This analysis includes description of overall structure, specification of how individual tasks are approached and finally a discussion of the solution. The task fully specified in chapter 5 was to generate a brief article that summarises what happened in a football match. To avoid ambiguity, throughout the article the word ”football” refers to a sport, which American English speakers call soccer. This is a first neat example of detailed aspects you need to be aware of when developing a NLG system - meaning of a word can change with different locations. Therefore, it is very important to know the target audience of the generated language.

1. Natural Language Generation

1.1 What is NLG?

The intuitive meaning of the Natural Language Generation (NLG) is rather obvious, unlike the definitions that usually vary. I will now present definitions of NLG by two different authors:

1. "Natural Language Generation (NLG) is the process by which thought is rendered into language."McDonald [2010]

2. "NLG is the subfield of artificial intelligence and computational linguistics that is concerned with the construction of computer systems that can produce understandable texts in English or other human languages from some underlying non-linguistic representation of information."Reiter and Dale [1997]

The definition (1) is much broader and less reliant on the specification of what is output and especially input of such a task, which is here defined as "thought". The problem of identifying the input has been discussed even earlier in McDonald [1993]: the broadness of source was compared to a human conversation, namely when the speaker starts deciding what to say. Then the input can be state of mind, current situation, speaker's intentions etc. These inputs are bordering on the impossible to classify and represent in a computer. The output is defined simply as language, without further specifying its representation.

On the other hand, the definition (2) defines output as understandable text, which implies the form of the result is written and additionally restricts the input to be non-linguistic data. However, in the article authors also mention that their survey is focused on written texts, but the principles could be also applied to generating spoken language, which implies the definition could be extended. Examples shown above, and especially the contrast of specifying what output or input can or can not be, explain why it is extremely hard to define the term NLG precisely.

Let us consider the problem of summarising a book into a brief description. Given this problem and definitions above, we can see that it satisfies only the second vague definition, since the input is purely linguistic. This kind of problem is referred to as text-to-text generation. An example of such a generation is extracting summary from a dialogue of a customer and customer service department described by Liu et al. [2019] or pun generation Ritchie [2005].

Similarly, the initial problem can be transformed into video-to-text generation by replacing a book with a movie. In this scenario the line blurs even more. A movie has two components: sound and the video itself. The video has implicitly some semantic meaning and is surely non-linguistic. However, the sound usually contains spoken language and therefore is linguistic. Take, for instance, a video showing a person named Mark pointing at an apple with the sound being "This is a pear.". The sound itself implies there is a pear, but the message could be summarised as Mark is lying or Mark is not able to recognize a fruit. Finding a correct summarization would be impossible with one of the components missing,

because they both affect the overall message, which makes deciding if video-to-text could be classified as NLG difficult. This logic of reasoning can be applied to other problems that vary in their initial inputs such as generating diagnostic reports from image (e.g. X-rays) by Zeng et al. [2020], which could be characterised as image-to-text generation.

In this thesis, we follow the definition by Reiter and Dale [1997] shown above. This means, in our understanding, NLG is a system creating text from non-linguistic data. This task is often referred to as data-to-text generation. Examples of definitions given throughout the article will fit this definition and the idea in general. Methods and approaches mentioned in this article may presumably be applied to any problem concerned with computational linguistics if it is suitable without a need to classify the problem as NLG.

1.2 Usage of NLG

The aim of the NLG is to generate documents, articles, reports, messages, emails, descriptions and other forms of texts in order to either reduce workload or to offer a reader a user-friendly interpretation of data in a given language. Various authors describe data-to-text implementations operating with different domains, input data and overall aims of the language:

- summarising data and creating reports
  - summarising statistics from a baseball match Puduppully et al. [2022]
  - summarising geo-referenced data such as map Thomas and Sripada [2007]
  - creating textual weather forecasts Sripada et al. [2014]
  - creating report of student's academic performance
- creating poetry (e.g. in Finnish by Hämäläinen et al. [2018])
- producing text to persuade reader something is good or bad Carenini and Moore [2006]

The most common usage of NLG is to summarise less readable data to a more convenient textual form regardless of the domain (sports, weather, geography, etc.). Even though the output is the same, the reason to apply the NLG system is different. Compare textual weather forecasts to reports of a student's academic performance. Weather reports are produced in order to enable the general public to find out information about weather since their ability to interpret meteorological raw data is probably lacking. Same reason appears when summarising baseball statistics and maps. On the other hand, there is no doubt that teachers and professors can correctly interpret grades from student's studies, but the problem is to do so quickly as well as obtaining the data in a well-readable layout. Generating textual summary automatically resolves both issues effectively at the same time.

The summaries and reports do vary in one more important aspect: the amount of the information and level of terminology. For instance, when creating a weather report for everybody to read, some information such as the type of rain or the

concrete amount of precipitation (mm) will not be mentioned in the output text and the information will be abbreviated to "heavy" rain since this is the information the reader is interested in. Contrastingly, when creating a medical report (e.g. from surgical procedure) the terms should be precise and technical (possibly in Latin) and the amount of information left out is close to minimum. To conclude, the target audience and their knowledge of the domain will alter the produced text substantially.

The overall aims of the language diverge significantly. In poetry the rhymed text filled with phrases calling forth emotions is used to achieve experience as powerful and captivating as possible when reading it. In summaries the aim is to convey the factual information to the reader in an easy and understandable way. Notice that this should be a significant aspect of choosing an approach for the problem. If the goal of the text is to inform, then text is highly recommended to be as simple as possible. On the other hand, if the aim is to captivate, as it is in newspaper articles, then creating simple dull sentences is not a suitable option.

This was the first glance into the aspects that are crucial when deciding how to approach a language generating problem. These aspects are mentioned many times across the text since the goal of this thesis is to present to the reader how to approach the NLG and the correct understanding is one of the key parts to develop a solid NLG system.

Is an NLG system good to use every time? Debate whether the software is worth creating is indeed viable and maybe a little underestimated since the articles usually do not analyse this feature elaborately. In the real world this is a question of economic resources. Wages and time saved on the human work that has been replaced with software must outweigh the cost of the creation and maintenance of the software. Easy examples of evidently beneficial usage of the NLG system is customer service emails generator. Even a simple inserting name to the start of the email, then stating the product, order number and a little survey of satisfaction with the product is quite a trivial email to generate, does the job and saves tons of company time. Possible example of a non-optimal usage is when text is not the most convenient form of data for the user to comprehend. Charts, tables, schemata, maps or pictures could be considerably easier to transfer the intentional message to the end user as these structures are usually attractive and intuitive.[1] To give you an example, imagine coloured map of a city and using a red line to highlight the path to the nearest tourist information centre in comparison with text composed of verbal instructions where to go. Obviously, unless the route is fairly trivial, a map is a better solution due to its simplicity and visuality.

[1] Combination of visual representation of data and text could also be the best alternative in a certain scenario.

2. NLG Tasks

Transforming non-linguistic data into grammatically correct sentences in a given language seems like a rather complicated problem. Therefore it is convenient to divide the initial problem into smaller tasks, which are easier to solve. This task structure is described by Reiter and Dale [1997] and it is widely used to cover every challenge a fundamental NLG problem should deal with:

- Content determination
- Discourse planning
- Sentence aggregation
- Lexicalization
- Referring expression generation
- Linguistic realisation

In this section we will discuss every task mentioned above. Note that approaches, which will be described later, may change this structure: there might be just a few changes by combining multiple steps and processing them simultaneously or the structure may be completely decomposed. Moreover, using data-driven methods is dominant approach and can be utilized across every task of the NLG. Their usage will be omitted in this section and further closely discussed in chapter 3 taking the uniqueness and dissimilarity of this approach into consideration.

The reason we describe every task individually is to highlight challenges that will arise along the way of creating the NLG system. Understanding these tasks is a crucial aspect to produce a well-built software regardless of the choice of the approach. To illustrate the problems we state numerous simple examples that should ease the process of fully recognizing the extent of issues related to each task.

2.1 Content determination

The goal of this task is to decide what information from input data should be included in the text. Usually the range of the input data is significantly larger than the amount of information we would actually transfer to the user. Naturally, this task is heavily influenced by the specifications of the assignment, namely domain, intention of the text and target audience. Consider the problem of creating a medical report from a complete blood count for a patient to read. This domain requires data preprocessing to recognize negative indicators from values contained in test results. In addition, a doctor or expert is needed to assist in order to interpret values correctly and set rules on how to identify diagnosis. The goal of the text is to describe the diagnosis in an understandable way and therefore using Latin or overly technical terminology should be avoided. However, if we take the doctor as a target audience, we now want to use the as precise expert

language as possible and probably change the content to present segments of the test results and not just the overall diagnosis to enable the doctor to better interpret marginal symptoms or flag values.

The result of the content determination is usually outputted as a set of preverbal messages, carrying semantic meaning of the statement. To carry all the information an implementation that can describe abstract concepts such relations between statements, entities or conditions is needed. This sub-task of creating suitable representation is usually domain-dependent. Since the important semantic information is mapped into some formal language, there is no need for (human) language to be specified, and therefore this task is language-independent. Concrete examples of formal representation language used to store these semantic attributes are for instance logical language, attribute-value matrices or graphs.

Example of the result of possible content determination is shown below in figure 2.1, where we would like to choose the content to report one simple message: a goal being scored in a football domain. We have two related (3) sets of attribute-value pairs: (1) is about a player and (2) contains goal statistics. Obviously some information in tables (1) and (2) are too specific (e.g. height of the player) for the message to convey and therefore redundant. Bold attribute-value pairs are highlighted as the ones to be present in the final text, creating preverbal message (4) in pseudocode, which is an output of content determination task. After performing the remaining NLG tasks possible result could look like (5).

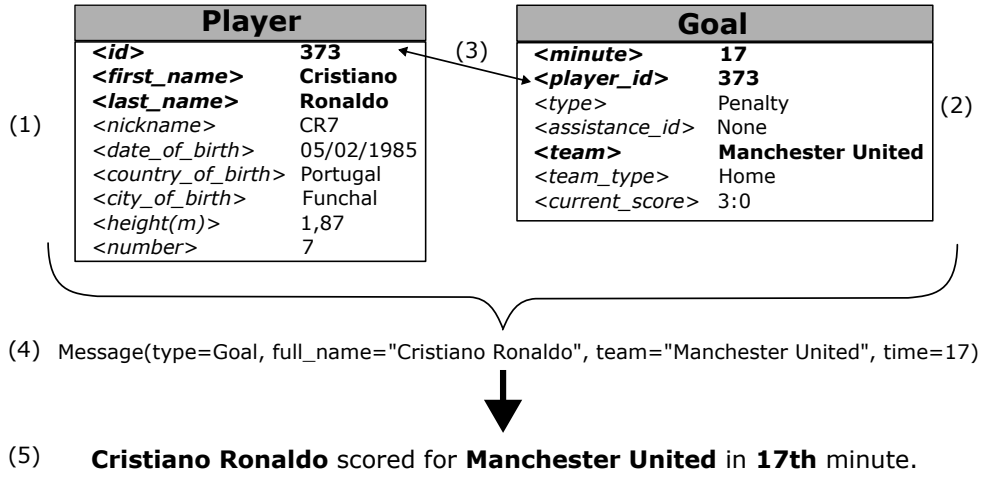


Figure 2.1: Process of content determination

2.2 Discourse planning

Previous part determines what messages will be transmitted to the reader and this part resolves the issue of the order in which the information is presented. This process is also referred to as text or document structuring. Selecting the right sequence of messages is crucial for text to accomplish its goal. Similarly we structure academic texts to logically ordered paragraphs, which present the topic in a way as understandable as possible for the reader to gain knowledge.

Similarly to content determination this task is highly domain-dependent as we have to know how to order messages. For instance, a medical report (as an example mentioned earlier) would likely display diagnoses and order decreasingly by how dangerous and life threatening they are. On the other hand, a report from a business meeting could start with a brief overview of achievements and goals and then with issues that were discussed ordered chronologically to allow the reader to follow the course of the meeting.

Human brain orders information to be conveyed in a speech intuitively, but the process as an algorithm itself is not quite trivial. Most of common method is to create rules based on the specific domain since the suitable structure heavily relies on the domain. Some researchers suggest using machine learning techniques for creating a uniform algorithm independent of the domain as seen in Dimitromanolaki and Androutsopoulos [2003].

Form of the output of discourse planning can differ. One possible option as described by Reiter and Dale [1997] is a tree structure. Leaves of the trees are messages and inner nodes describe specifics of their function in a sentence. This may seem like an unnecessary complicated solution when clustering messages to be said in one sentence can be just an array of messages. The benefit of the tree structure is the amount of information we can store along the messages including constraints under which the message can be said, relations between them and their overall structure.

2.3 Sentence Aggregation

The cardinality of the relation message and sentence is rarely one-to-one. Usually multiple messages are formed into one sentence. This process is called sentence aggregation and it is fundamental for generating text that is readable and flows well. To clarify we provide set of verbal messages:

1. *Peter bought an apple.*
2. *Peter bought a banana.*
3. *Anne did not buy anything.*

This set of sentences is clearly non-optimal and can be aggregated in two steps as follows:[1]

4. *Peter bought an apple and a banana. Anne did not buy anything.*
5. *Peter bought an apple and a banana, whereas Anne did not buy anything.*

We can notice two types of aggregation leading to the optimal sentence (5). Aggregation of:

- **constituents** - Constituents that have equal syntactic importance can be aggregated using suitable coordinating conjunctions expressing their relation. Take example sentences (1) and (2). *Apple* and *banana* are both

[1]Naturally, no such concept as "optimal" sentence exists. The optimum in this case is to express the information in a sentence that would likely occur in a spoken human language and also would appear fluid and natural.

items *Peter* bought so their semantic meaning is identical. Therefore they can be aggregated via cumulative conjunction *and* creating a new noun phrase in the result sentence (4) *an apple and a banana*. Another example of cumulative conjunctions is *both ... and* or *as well as*.

- **sentences** - Sentences can be aggregated as well using coordinators as seen in the result sentence, which was created by inserting an adversative conjunction in between sentences in example (4) to express opposition. This contrast can be expressed by other words like *but*, *yet*, *while*, etc. More relations can be expressed when aggregating sentences using another kinds of coordinating conjunctions: *alternative* (*or*, *either ...or*, *nor*) to express two or more alternatives and *illative* (*for*, *so*) to express interference or consequence.

One more type of aggregation can occur based on explicit hand-crafted domain-based rules. Take these three preverbal messages from football domain reporting a goal, which are similar to the one as used in figure 2.1-(4) (Manchester United shortened to MU):

6. *(type=Goal, full_name="Cristiano Ronaldo", team = "MU", time = 4)*
7. *(type=Goal, full_name="Cristiano Ronaldo", team = "MU", time = 8)*
8. *(type=Goal, full_name="Cristiano Ronaldo", team = "MU", time = 14)*

Surely realising messages (6), (7) and (8) as three different sentences would not create fluid and natural results as sentences would vary only in time of the goal. Since three goals in football form a so called hat-trick we could aggregate messages in one sentence:

Cristiano Ronaldo completes hat-trick for MU in under 15 minutes.

This aggregation realises the messages (6, 7, 8) beautifully as the result is well-formed and natural. Notice that the aggregation happened also in expressing the time of the goals: instead of mentioning three different timestamps the time is summarised as "under 15 minutes" highlighting the fact that this rare figure was achieved in a short amount of time. As described in the previous section, domain knowledge is necessary to decide what is "long", "short" or "average" (and therefore not worth mentioning) amount of time for an event to happen.

Note that these aggregations are simple for humans, but to perform them in a NLG we need some semantic knowledge and relations of the sentences (or constituents). The easiest approach is to define domain-specific constraints when to perform aggregation. Defining complex domain-independent rules and universal representation of relations is rather a difficult task and nowadays often solved using data-driven methods, which are described later in chapter 3.

Furthermore, the idea that the more aggregations we perform the better the final text is wrong. Sometimes slowing down the flow of information by fracturing the message into smaller individual sentences is useful in order to produce more understandable text. Overloading sentences can often result in less fluency as the more information is conveyed in one sentence the harder it is for a reader to follow. Barzilay and Lapata [2006] are perceiving this as a linear programming problem

where similarity is classified for each pair of database entries. Using this similarity, transitivity and global constraints (e.g. maximum number of aggregation across the document) they find an optimal solution.

2.4 Lexicalization

After performing discourse planning and sentence aggregation the preverbal messages are in a correct order and they contain suitably aggregated information. Goal of this task is to create mapping from these messages to specific expressions in a given human language. This task is the first that is language-dependent. There are two main problems associated with lexicalization. Firstly, the amount of combinations of how to narrate a message is enormous, only restricted to those that fit into the given context. And secondly, transformation of concept into a word (or more words) is very abstract and interferes with many layers of the language (semantics, phonetics and pragmatics) and therefore choosing a suitable expression is rather difficult. This transformation is not even easy for humans. Imagine an essay contest in grammar school with a given topic of the essay. If the transformation was easy and had only one solution, the contest would not exist as essays would be identical. In fact, the perspective and overall understanding of the topic, style of describing one's point of view and finally even choosing words to present the idea is partly what distinguishes us as people.

Another factor is the target audience and the overall goal of the language. If the target audience is educated on the matter then using adequate technical terminology is reasonable. Contrastingly, for low-skilled readers all terminology must be explained in an easy way and the content of the text should be more about overall ideas rather than about specific concepts.

Trivial approach to this task is to hand-craft pairings of a word or a whole phrase and a concept in a message. This solution results in monotonic outputs as the aspect of choice is missing. Slight improvement would be to add more semantically similar options for each item. However, this can cause problems. First of them is how to decide, which possibility is the best. Second is the possible non-viable combinations of words together. One example, that may be not visible on the first glance, is generation of adjectives interpreting numbers. For example, take a person with height 185 cm. If it was a man, the height is "average", while a woman could be described as "tall". Therefore semantic background and suitable comparison need to be taken into account. What is more, combinations of chosen phrases may result in non-realisable or simply weird expressions.

Due to the vagueness and coherence of the process, NLG systems combine lexicalization, REG and linguistic realisation under one operation called surface realisation or tactical part of the process.

2.5 Referring expression generation

Referring expression generation (REG) is a process, when you choose words to express domain entities or other constituents of the message. Naturally, utilising one noun phrase for one specific entity, which is used more than once in a short amount of text, results in less readable and fluid text. On the contrary, there

is a limit to how many such expressions we can generate since a reader needs to identify the entity correctly. Ambiguity is a highly unwanted effect since the information that needs to be conveyed may differ from its actual language semantic meaning.

To fully understand the challenges and also possible solutions for REG here is an example of sentence where we would like to lexicalize its subject represented as an entity in pseudocode:

(entity=Country, name="Czech Republic") has a population of 10,3 million.

This particular country can be lexically expressed in this sentence for example as:

1. *Czech Republic*
2. *Czechia*
3. *Country nicknamed "The heart of Europe"*
4. *Bohemia, Moravia and Czech Silesia*
5. *Country in the central Europe*
6. *Country which borders Germany, Poland, Slovakia and Austria*
7. *Beautiful rustic country in the heart of Europe*
8. *It*
9. *This country*

Notice the linguistic techniques we used to express this subject:

- **Entity name** - Using the name of the entity is a trivial solution and works fine as seen example (1).
- **Synonyms** - Using a synonym or a different name having the identical semantic meaning for the entity as shown in example (2) and (3).
- **Descriptive transcription** - using the knowledge of physical appearance, characteristics or its location we can describe an entity without any need of using its initial name. Examples (5), (6) and (7) are all using the location in Europe. Although expression (6) identifies Czechia unambiguously, the description of the location may be too specific for a less-educated reader (or possibly for a reader living outside of Europe), who has no idea where Germany is. The information that the country is situated in Europe is then sufficient. Therefore the target reader, his knowledge about a topic and also the purpose of the text are important even in this task. This technique is also prone to ambiguity as seen in example (5): reader should already know what country is described in the text to use this expression since more countries can be characterised as "central European".

- **Definite descriptions** - The expression can be enriched by adding valid adjectives, adverbs or other linguistic structures to further specify the object as seen in example (7) where adjectives rustic and beautiful describe the country even a little bit more creating enriched and complex noun phrase.
- **Pronouns** - In a human language pronouns are often used to represent entities (e.g. "I have seen him." or examples (8), (9)). Using pronouns correctly can help to improve readability of the text and also minimise the obvious flags of computer-generated text. The main obstacle to overcome is when to use pronouns. Sometimes usage of a pronoun can arise from context, sometimes if there is absolutely certainty that everyone knows what the pronoun is referring to: those examples are hard to deal with and usually handled explicitly. Usual approach is to use pronoun if the entity was mentioned in a previous sentence under the condition the entity was the only constituent the pronoun could refer to.

How to approach REG depends also on repetition of the entities in the text and the final text variability. In a domain where identifying the entity unambiguously is primary (e.g. city in air travel) the usage of REG is even harmful. For instance, expressing "New York, USA" as "The city that never sleeps" in the air travel domain. To the contrary, expressing an entity identically multiple times in a short span of prosaic text eventuates in dull, plain and stereotypical language.

2.6 Linguistic realisation

This task transforms every constituent to form a grammatically and syntactically well-built sentence. Example of a struggle when building even a simple noun phrase is:

1. *(entity=Animal, name ="dog", count=1)* $\rightarrow$ *one dog*
2. *(entity=Animal, name="dog", count=23)* $\rightarrow$ *23 dogs*
3. *(entity=Animal, name="mouse", count=2)* $\rightarrow$ *two mice*
4. *(entity=Animal, name="fish", count=1000)* $\rightarrow$ *thousand fish*

Trivial and also naive solution for this simple noun phrase building can be to append morpheme "s" to the name of the animal if the count is more than one. As you can see in example (3) and (4) this solution can work only for animals that have regular plurals. In addition, the count itself is recommended to be expressed by a word and not by numeral if the number is either a small integer (1-10) as seen in the comparison of examples (1) and (2). Same rule applies as well for a well-known rounded number (hundred, thousand, billion, etc.) shown in (4).

One, so far omitted important, aspect is the principles of morphology and syntax of the language. Concept of appending morpheme "s" to express plural might not be so easily transferable into different languages. In Slavic languages morphemes to express plural differ and also can be infixed, meaning they could be inserted into the word stem instead of using a suffix. In Czech, a word for

dog is *pes*. Then realising their number would look like: *1 pes, 2 psi, 5 psů*. To further complicate the situation, Slavic languages are fusional. That means a single morpheme carries multiple grammatical, syntactic or semantic meanings. In Czech, agreement between the grammatical case and the noun is resolved as well with an infix morpheme. Here are three examples in Czech using different grammatical cases along with translation from English:

- (case: nominative) *two* ***dogs*** → *dva* ***psi***
- (case: genitive) *without two* ***dogs*** → *bez dvou* ***psů***
- (case: instrumental) *with two* ***dogs*** → *s dvěma* ***psy***

Notice that not only the noun, but the word for number 2 (*dva*) as well are subject to the linguistic transformation as the infix morpheme carries both the agreement with case and the plural.

We have mentioned typology according to morphology (fusional language). Another type is analytic languages (e.g. Vietnamese), where every single word is exactly one morpheme. Lastly, the complete opposite is polysynthetic languages (e.g. Inuit languages), where one word is constructed by combining plenty of morphemes representing a whole sentence. In central Nunavut Inuktitut *Tusaatsiarunnanngittualuujunga* means *I cannot hear very well*.

The extent of language influence is huge even in the semantic part as well. To demonstrate, imagine generating a sentence, in which the action will happen in the future. This may seem like a trivial problem in English since we may just use the word "will" (or other modals). However, English, unlike Romance languages (French, Spanish, Italian and more) does not have future grammatical tense. Some languages are even tenseless, e.g. Tokelauan spoken in American Samoa (Polynesia). This section just illustrated the need for the knowledge and principles of the language during the linguistic realisation.

Realisation has quite an extensive magnitude caused by the non-trivial goal of correctly morphologically and syntactically (as well as semantically) expressing each of the lexical items in the sentence. This process requires adding auxiliary words (prepositions, verbs, etc.), handling agreements, ordering, inserting punctuation and other similar transformations all in order to present the language not only factually, but even grammatically right. Due to the complexity of the task multiple approaches exist. One of them is using templates.

Templates are hand-crafted using fixed lexical items and attributes substituted in the template. Preverbal message (1) is assigned to a template (2) and three variables are then substituted with values creating the target sentence (3).

1. preverbal message → Message(type=Goal, full_name = "Cristiano Ronaldo", team = "Manchester United", time = 17)
2. template → **$full_name** *scored for* **$team** *in* **$minute***th minute*.
3. result → **Cristiano Ronaldo** *scored for* **Manchester United** *in* **17***th minute*.

The advantage of this approach is simplicity and prevention of grammatical errors considering that we have full control of what the fixed segments are and any unwanted error is highly improbable. The disadvantages prevail. First of all, applying templates could be only feasible in well-defined low-volume domains as entities must be have easy-to-text-interpretation. Another reason is that creating templates is time-consuming. And most importantly, the variation of the output is low as the immutable parts of the text generate very limited output. In addition, the template approach for more complicated languages tends to struggle, because the constituents usually depend on each other (e.g. agreement, auxiliary words, etc.) creating requirements that are hard to fulfil.

However, templates are extremely practical when the target output is expected to be simple and rarely changing. Great example is generating spoken announcements in the transportation domain e.g. departures of flights on the airport, where the template could look like: *The departure of flight number $number from $destination_from to $destination_to will be slightly delayed.* → *The departure of flight number FD-2018 from Rome to Paris will be slightly delayed.* Results are admittedly blunt in terms of language richness, but they are factually correct, clear and easy to comprehend, which was the initial purpose.

Other approaches are more complicated in order to outperform templates in a range of expressions. First of them is building a grammar for the natural language. This approach relies on thorough knowledge and examination of the language behaviour and principles offering domain-independent solutions that can be applied to a different NLG system performing linguistic realisation. The advantage of this approach is definitely the domain independence of the realiser as well as its variety in produced output. Disadvantage is that building such grammar is labour-expensive and the disability to select the best possible result. All generated sentences will be correct, but the choice, which one is the optimal one is beyond grammar's reach.

Secondly, data-driven (further described chapter 3)methods can be applied in this task too. However their usage can vary greatly. Both approaches above can be enhanced by these stochastic methods since we can reduce manual workload. Templates can be automatically extracted from training corpus (more about corpora in chapter 4). Similarly, hand-crafted grammars can be created automatically from corpora. Also, linguistic approaches can be avoided by using these methods. However, fusing traditional linguistic approaches and the power of statistical approach can result in a well-performing system. For instance, using hand-crafted grammar in combination with stochastic methods to resolve the optimal-among-the-correct-ones issue.

3. NLG approaches

So far we have covered what individual tasks we need to cover in our solution without suggesting a proper structure, which organises these tasks together and creates a compact implementation. There are two main approaches:

- **Modular architecture** - The basic idea is to split the program into parts that handle the NLG problem task-by-task. The division of the tasks is not always clear and therefore the idea of this approach is joining closely related tasks into an individual unit: module. These modules are then linked with a pipeline offering systematic and well-structured architecture.
- **Global approach** - The tasks are fully deconstructed and the problem is solved globally in order to get rid of the limitations associated with modular architecture. Such NLG systems then process smaller segments of NLG tasks alternately in order to make the optimal choice each time with every context state and next step available. Utilising this freedom is the core aspect for success of this approach. Two methods realising decomposed architecture will be discussed: Planning and data-driven methods.

This division is definitely not strict as NLG systems offer a wide range of architectures and their combinations. Data-driven (or statistical, stochastic) methods are a widely discussed topic among the NLG community as their popularity grows and their results are recently getting better, often outperforming more traditional procedures. Note that statistical methods can be incorporated inside a modular architecture to resolve one or more tasks. Similarly, you can take the global approach and avoid data-driven methods completely. Also grouping many arbitrary tasks together and approaching them differently is a possibility. This section is about describing such approaches in detail.

3.1 Modular architecture

This approach was described by Reiter and Dale [1997] and became almost a standard for a long time. However, nowadays stochastic and data-driven methods are prioritised over this approach (further discussed later in Chapter). The idea is to construct a module for each NLG task and link those modules via a one-way pipeline. Hence, output of content determination is an input for discourse planning and so forth. But some of the tasks are closely related and therefore it is efficient to assign those tasks to one module. Accustomed layout by Reiter and Dale [1997] of modules is:

- Text (document) planner → content determination + discourse planning
  = *"what to say"*
- Sentence planner → sentence aggregation + lexicalization + REG
  = *"how to say it"*
- Linguistic realiser
  = *"saying it correctly"*

Text planner determines the content of the text as well as the order, in which we present the content to the reader. This part of the NLG process is also referred to as strategic generation. Sentence planner transforms messages from text planner to a lexicalized expression. And finally linguistic realiser combines these expressions with regard to syntactic and grammatical rules of given language. Choices made during sentence planning and linguistic realisation are often together called, on the other hand, tactical. Clear distinction between tactical and strategic will be more important later when discussing data-driven methods. The main advantage of this 3-module is the structure itself, which is easy to follow and understand. In addition, offering clear division of what is each module's obligations as well as what issue they don't resolve. Simplicity results in accessible and well-structured code. Moreover, it is easier to change a minor functionality of the program since classifying where to change the code is intuitive. The same logic applies for upgrades: one or more tasks can be approached completely differently than the rest of the code using other methods (e.g. statistical) by keeping the input and output structure.

The core drawback of the approach arises from the same aspect which was mentioned above as an advantage: well defined clear systematic structure. This brings certain limitations. Once we decide sentence order and content we have no chance of changing it later and so the choices in the early stages may later result in an unsolvable issue. Imagine generating a sentence with a limited maximum number of characters. In a text planner we have chosen the content of the sentence, but even when we do every possible combination of lexicalization of the sentence the number of characters still exceeds the upper bound. So what now?! The solution would be to retroactively change the content of the sentence and drop part of the initial information, but this is not possible since the pipeline is one sided. Of course this problem could be bypassed by making the pipeline go backwards, but this would completely break the point of modular approach. The clear line of division among modules would disappear and modules' functionality and objectives would suddenly overlap. To put it simply, assembling the structure of the text before knowing linguistic resources may result in incorrect, ambiguous or bizzare expressions. Therefore other approaches are usually based on breaking the structure and skipping to different tasks depending on the state of the development and the constraints that arise along the process.

3.2 Planning approach

In order to produce a text in a decent quality many decisions must be made resulting in various alternatives. Similarly as described in Fikes and Nilsson [1971] where the idea is to find a universal robot solver for the world model represented as first-order predicate formulas. The broadness and vagueness of the process of starting with various preconditions and getting to the desired result lead to another approach, which highly differs from previous two mentioned approaches: planning.

Planning problem (as well as planning approach in general) is described by Gatt and Krahmer [2018] as "the process of identifying a sequence of one or more actions to satisfy a particular goal". Actions are then described as preconditions and their effect after applying them. In the terms of NLG, the goal is to convey

the message along with its aim (persuade, inform, captive, ...) to the target person. The actions then have constraints under which they can be performed and the effect is the change to the current context all leading to the desired goal of the language. The main idea is to create formalism that does not rely on a strict structure of the NLG system and available solutions to alternate between different NLG tasks with the recognition of both current state and effects of the chosen actions all in order to create the best possible language and broaden the limits of pure modular approach.

3.3 Data-driven approach

Unlike both mentioned approaches above, data-driven methods do not define the architecture of the NLG process. Modular approach states three-module architecture connected via one-way pipeline, whereas the main idea behind planning is not having predetermined architecture at all and approaching the process globally with a well-build formalism enabling freedom for combining actions in. Data-driven methods can be applied regardless of the choice of the architecture. Meaning that stochastic methods can be used in a global approach, single NLG task or certain bigger subsections like strategic or tactical parts, for instance.

As the name implies, data-driven methods crucially rely on data, which consist of inputs and corresponding outputs. Using statistical or probabilistic principles of comparing our current state of the NLG process to a similar state in the data ensures making choices similar to those in the training dataset: in the field of linguistics this date is referred to as corpus. Since these methods can be applied on even smaller segments of the generation data can contain processed input or output data in various internal representations, which appear during the process and not just the initial and final stage. For example, when computing sentence planner inputs of our testing data are preverbal messages and outputs are lexicalized texts.

The very first obstacle that arises when performing these methods is the acquisition of the input-output data, because the data must fulfil some requirement. The amount of the dataset should be big enough to ensure the validity and overall principal of the statistical and probabilistic approach. The dataset must not only satisfy the requirement on the quantity, but certain variety must be ensured as monotonic data tend to give one-sided and misleading results.

Acquiring data itself can be tricky. Easiest scenario when an already established corpus for a specific domain (e.g. weather forecasting, hotel and restaurant recommendation, sport reports and more) exists. These corpora are well-built, but on the other hand useful only when working with the same domain. The reason for that these corpora may be available is firstly because their usage is common and secondly for their input data simplicity. Working with a less-common domain with larger range, types or complexity of data will result probably in the absence of a viable corpus and therefore building the corpus is another problem appended to the NLG process. To overcome this problem we can either build a new corpus from scratch or exploit more stochastic methods that automatically align input with outputs. Then again one disadvantage is that the data are heavily domain-specific. The alignment of the input to the segments of output is crucial for most of the methods except deep neural networks and other

machine-learning methods. These methods are recently becoming dominant in certain subfields of NLG such as image-to-text generation.

After assuming we have acquired data for the full-ranged NLG process, we can classify a stochastic approach based on the overall architecture. One group approaches the problem globally and completely decomposes the modular approach in order to both avoid error propagation and allow the software to make decisions freely across multiple tasks and different stages of generation. In opposition, the second group upholds at least the division between tactical and strategic choices.

As mentioned in the beginning of this section, the data-driven methods can be applied only to process one of the NLG task and not a whole generation in general. So far we have described how these data-driven methods work in general without specifying these methods in detail. We state some examples to further illustrate usage of data-driven approach with different specific methods and range of NLG process they cover:

1. **stochastic process** - Work of Ratnaparkhi [2000] is a nice first example to introduce data-driven methods, because he described three systems (NLG1-3) along with their comparison. Systems are used for tactical generation (semantic context is provided in a corpus given attribute-value pairs aligned with textual outputs) in an air travel domain. First system (NLG1), for given attributes, simply chooses a template with the highest number of occurrences in the training data. Second system uses a maximum entropy probabilistic model to predict the best proceeding word taking the already generated and also the attributes yet to be generated into account. However, the dependency of the words in language may not come from their order. Therefore NLG3 predicts the best words based on their syntactic relations represented by a tree. Along with the system's description Ratnaparkhi [2000] offers their results in terms of correctness. Both NLG2 and NLG3 heavily outperformed NLG1. Furthermore, NLG3 was performing slightly more accurately than NLG2. Ratnaparkhi [2000] states that both NLG2 and NLG3 can be used in other domains as well, but the complexity of the domain must be somewhat similar to the air travel (meaning quite low). Implicitly domain annotated data must be provided to further exploit NLG2/3 systems.

2. **classification** - Duboue and McKeown [2003] used classification process in their system for automatic content determination illustrated on biography generation problem. System is provided with initial data and target texts. Algorithm starts with clustering the semantic data (e.g. by age) and matching segments of the output to the pieces of input. This forms a solid base for the process of creating content selection rules using the binary classifier for each attribute of input data whether the attribute should be mentioned or not.

3. **optimisation** - The article of Marciniak and Strube [2005] researches the optimisation process in Natural Language Processing (NLP) approached as an integer linear programming problem. They use this approach even in a field of NLG when generating textual route directions. The global approach results in the elimination of error propagation and has better

overall results as a consequence according to Marciniak and Strube [2005]. Summary of the specifics such as the metric that should be minimized during linear programming is further described, for instance, in Gatt and Krahmer [2018]).

4. **probabilistic context-free grammar (PCFG) and parsing** - The idea of mixing the NLG tasks together is further exploit by Konstas and Lapata [2013] as they call the of every single task separately "greedy". The need for domain-specific approach is here eliminated as this work is concerned with concept-to-text generation. They use input to model a PCFG and then use the stochastic methods to acquire the best word sentence satisfying the grammar. Such a process can be viewed as an opposite to semantic parsing. This system was tested on three different domains: sportscasting, weather forecasting and air travel query generation. Performance results were described as same or even better to the methods known at the time.

Note that these four examples not only show various specific statistical methods, but they illustrate other nuances as well. Take the range of the process they cover for instance. Examples (4) and (3) cover end-to-end process, contrastingly example (2) covers only the tactical part and example (1) only covers content determination. Moreover (1) and (2) keep the strategic and tactical division unlike (4) and (3). Domains differ as well, especially in example (4), in which domain does not have to be specified.

These examples share two more similarities except the implicit statistical approach. Firstly, they rely heavily on the testing data and especially their alignment of input and output. Secondly, their results are somewhat superior to other hand-engineered systems. Often hand-crafting a NLG system relies heavily on the domain and lacks portability and certain variability of the output (respectively achieving the variability by hand is the more tedious job the more variable the output should be). NLG systems grounded on statistics are robust and a task of data acquisition is added to the end-to-end solution. This is counterbalanced by overall better results and more portability as example (4) is domain-independent. These methods along with the solid linguistic foundation are nowadays dominant since the robustness is not big enough to be uncomputable with modern computers and the availability (or the opportunity to compute the) and amount of testing data is much better.

4. Process of developing an NLG system

The process of developing a text-producing software is complex and requires organised preparation in order to deliver a quality result. This paper so far covered separate segments of the NLG accompanied by numerous examples to illustrate concrete problems. Now we suggest how to order key parts of the development:

1. Requirements analysis
2. Corpus building
3. Choosing suitable architecture and approaches
4. Implementation of the NLG system
5. Evaluation

Steps (1), (2) and (5) are further discussed in separate sections below. Steps (3) and (4) were already described in detail in chapter 2 and chapter 3. Both steps (2) and (5) are skippable and not required, but their implementation can further enrich the final solution.

4.1 Requirements analysis

First step of developing the NLG system is to carefully analyse fundamental aspects that will establish the approach and overall functioning of the program. We state six such aspects:

- Input data
- Expected output
- Target language
- Goal of the language
- Target audience
- Usage of already existing tools

Examples in the article were created to demonstrate the various effects these aspects could have. This section will briefly summarise nuances that can occur.

Input data can have different inner representations: table, database, graphs, images, etc. Furthermore data can be in its initial state and probably will require some preprocessing to, for instance, extract computable information or transform data into better structured and more suitable other representation. Some representations can result in creating non-trivial tools to extract needed information for the generation, e.g. image requires image recognition software.

Expected output can be specified in corpus (described below). Knowledge of desired output is key to analyse the full extent of other aspects.

Target language can as well influence the approach greatly, especially the solution for the tactical part of the program. In the section about linguistic realisation (chapter 2) we have covered some differences between languages. Naturally, deeper linguistic analysis is required to fully recognize the specifics of the language and how to approach lexicalization, REG and realisation. Moreover, the NLG system can generate identical messages in multiple different languages and then the approach for the solution can be restricted to only those that strictly divide tactical and strategic choices.

Goal of the language is very important as seen in examples throughout the article. For instance, producing a routine text with formal requirements will be approached differently than generating an eye-catching book teaser.

Target audience has a couple of characteristics: age, literacy rate, level of education or knowledge of the domain and topic. These details should be taken into consideration in order to deliver the message to the reader in the most convenient and understandable way possible.

Lastly, a great number of **already existing tools** has been developed and it is a possibility to incorporate these tools into newly constructed solution. Examples can be domain specific corpora or specific language realisers. Utilising these already established resources can upgrade the solution. However, the negative impact of such usage can be making ponderous transformations to create input in correct form that is required by the incorporated system as well as accommodating the approach to the tool.

4.2 Corpora building

As we mentioned above, requirement factors can greatly influence methods and our overall approach to the problem. Communicating these specifics can be tricky when only one side has an insight into computational linguistics. The most suggested method is to create a corpus, which is a collection of inputs and corresponding text outputs. This process ensures a clear definition of expectations of what the output will look like and prevents misunderstanding. This method is useful even when developing software independently without any specific party, that would require any specifics. Corpus is then built to our own expectations creating a solid base for the process of approach analysis and implementation of the NLG system in general.

Additionally, corpus is necessary when applying statistical methods since they rely completely on the content of the corpus and the correlation in-between. Lastly, corpus does not have to be composed of end-to-end NLG pairs, but rather contain inputs and corresponding outputs for a smaller part of the NLG (one or more tasks) in order to create a foundation for stochastic methods to resolve the problem. Inputs and outputs can then vary greatly in the inner representation as for instance inputs for content determination and REG will surely differ. What is more, since data structures alter even for the same NLG task solution the corpus should be built (or transformed) so that the input and output fit the specific solution's representations.

Hand-building a corpus from scratch should be supervised and consulted with

a domain expert (e.g. for creating medical reports the doctor should participate in creating example texts) in order to achieve the best result possible. For corresponding input an example output should be written by a domain expert. The result of this process is called an initial corpus. The developer should now make a revision of the initial corpus to guarantee that the NLG system can deliver the expected text, because not every input and corresponding output must necessarily be feasible. Firstly, the output text can be improved (e.g. when acquiring texts from existing one). Secondly, the information that is contained in the sentence may not be present nor computable from the given input data.[1] This is a critical part of the development as there is no way to resolve this problem every time and it is highly application-dependent. Common solutions are to extend the input data or remove unavailable information from the text and create a new version avoiding the information. Similarly, a compromise of finding hand-formed rules when to convey the information may be the solution. After all the necessary changes to the initial corpus were made, the corpus is now composed strictly of well-built and agreed upon example texts, where every information needed to its generation is contained in a data directly or it can be computed from given data. This finalised corpus is called target text corpus.

What should a good corpus look like? Corpus should be comprehensive and offer a wide range of pairs input-output. Edge cases, exceptions or less unusual texts should be incorporated in the corpus as well as average text to produce. Naturally, the number of normal texts to produce should be significantly higher than the number of rare examples to indicate the correct ratio. Corpus should be exhaustive in terms of expectations: once the corpus is finished, adding functionality explicitly is very complicated and is likely to change the overall structure, depending on its complexity.

Described process of hand-building corpus is highly labour-expensive and therefore usually domain-dependent systems for automatic acquisition of the data are developed lately further exploiting the strengths of statistical methods in combination with the amount of accessible data. Some NLG problems can find better results when extracting output texts from already existing "approved" texts. For example, when generating a short weather forecast it would be ineffective to create newly-written texts. Much better approach would be to extract forecasts from the most popular weather websites and take those as expected outputs. Benefits of this approach are reduced time spent acquiring example outputs and also ensured quality of the text since using popular websites. Some downsides could arise when applying this approach: acquiring this data can contradict with copyright law and range of the input data can differ resulting in unavailable knowledge. Also the output text of our new implemented NLG system (when done optimally) will produce similar texts as those popular websites and therefore there is no reason why our forecast should become more read and popular than the already existing ones. This approach is suitable when the domain content will probably be very similar, reducing the amount of knowledge that may be unavailable. On top of that, the number of available output texts must be high to make use of the

[1]The information can also be somehow contained in the initial data, but building tools for its extraction would be insanely time consuming or the time complexity of the extraction would be high and therefore not possible: this is up to the developer to analyse and determine data transformations within his reach.

stochastic approach. Such domains are weather forecasts, sport reports or the (air) travel domain and more.

To summarise, building a corpus is not obligatory, but highly recommended practice to ease the process of developing the NLG system, especially finding requirements. Precise target corpus primarily precedes the problem that the quality of the developed system is insufficient as the user knows exactly what the output will look like and match his expectations perfectly. When applying statistical methods, a well-defined and comprehensive corpus becomes a necessity since the stochastic approach relies crucially on the power of data. Lastly, the corpus is a fine tool even for the developer himself: analysis of each record of the corpus results in an outline of the challenges to overcome and gives a basic idea how the particular NLG problem should be approached.

4.3 Evaluation

After NLG yields the textual result an intuitive action would be to rate it and then reflect the approach and implementation based on the rating. From a practical point of view this step can be processed by a user or client who expects a certain quality and decides whether the quality is matched or not. From an academic point of view, this task is about deciding how to measure what is the "optimal" output as described in footnote 1. Computer science problems usually contain a metric, which easily decides what solution is better. For instance, graph problems usually have an optimal solution to the problem and algorithms are compared based on time complexity of finding this optimal solution.[2] However, in linguistics the comparison metric is not obvious, but rather complicated.

This can be demonstrated in a school-leaving essay. Deciding about the grade will likely be uniform since the base is some general criteria that have to be matched. However, how to decide what work was the best (if the goal was to pick the best essay). Surely, we can restrict ourselves to only the theses that were graded with the best grade, but how to choose among them? To summarise, in linguistics it is trivial for a human to decide if the result is "good" or "bad", but it is difficult to pick the best among the "good" ones. "Bad" results can be grammatically incorrect, factually wrong or the communicative goal was not achieved. This process is even harder for computers and therefore a human element is often required to rate the results.

It seems that the measure tools differ very much from what to measure to how to to measure it. Readability, fluency, and correctness can be viable attributes rating the linguistic appearance of the text. What information is present in the text, how it is presented to the reader, whether the information is relevant and if the communicative goal was achieved are attributes to grade content of the text and the overall vibe. Grading these attributes can be then approached alternatively as well: by grading on discrete or continuous scale, ordering texts or simply picking the best text in a given aspect. The metric should be designed to reflect important aspects for the given texts.

The uncertainty of this step is predictable as the importance and consequences

[2]Exceptions are probabilistic and approximation algorithms that find acceptable (compared to the optimal) solution in a faster time

of small nuances in the texts are interpreted subjectively. Gatt and Krahmer [2018] recommend aiming for diversity in the approach to yield a wide range of results and then decide the best final approach based on not only the results, but on the correlation between them as well.

5. Implementation

The aim of the thesis was not only to get the basic understanding of an NLG process, but also to try to create the implementation in a language that is widely used around the world: Python. The goal was to create an end-to-end NLG system, that takes non-processed raw data about a football match as an input and creates an article in Czech language that summarises the course of the match. In this chapter we will take a closer look at the overall functioning of the football articles generating NLG system.

5.1 Requirements and initial goal

As mentioned in the introduction of this section, the expected output was simply a readable Czech text summary while attempting to produce sentences that are not identical, resulting in a less-monotonic text. Since the expected output was defined somewhat vague and quality-wise uncertain, there is no stress on the definition of the goal of the language as well as target audience. However, the text quality, especially in the variability of expressions, used structures and overall richness, is insufficient to truly satisfy the definition of an (newspaper) article.

Since NLG is a complex process we used one already existing tool for linguistic realisation - *Genja API* by Geneea. Without this system the output language would not be grammatically correct and therefore not satisfying the readability requirement. As discussed in 2.6 this task has no easy way of implementing manually, especially not for a Czech language that belongs to the most complicated natural languages (for reasons that are described in 2.6 as well). *Genja API* usage skips this task focusing on the rest of the NLG tasks.

In chapter 4 we proposed an outline to a requirements analysis. This chapter tries to follow this structure discussing and describing the specifics of each segment. Unluckily, in order to present the closely-connected requirements and their consequences in a organised way, we may modify the structure slightly.

5.2 Input data

The dataset was provided by the company Livesport s.r.o. (see `https://www.livesport.cz/`), which I would like to thank. Therefore the entire dataset can not be shared. However, one example match is attached to illustrate program functioning. Initial dataset is composed of every match of one Czech First League season. Every match is represented as a JSON file storing both general information about the match (teams, venue, attendance, line-ups, etc.) and course of key events of the matches (goals, substitutions, cards, etc.).

General information about the match consists of two participating teams, starting time of the match, tournament information (here Czech First League, season 2018/2019), information about venue, score, winner, stage of the game (e.g. finished/delayed) and line-up. Line-up consists of every player of the team divided into groups according to their status (e.g. injured, benched, initial line-up, etc.) along with other information like home country, number and more.

Course of events is represented as a list of incidents. Incidents have attributes like id, time, participant, type and so on. Also, incidents have attribute *parentId*, which further specifies the incident. For instance incident type *Penalty Kick* has empty *parentId*, however, in the list of incidents there is an incident with the type of *Penalty Scored* or *Penalty Missed* that has *parentId* identical to the corresponding *Penalty Kick* incident *Id*.

Detailed examples of the representation of the input data are shown later in figure 5.3 and figure 5.4.

5.3 Approach

For this particular NLG problem we have chosen modular architecture as described in 3.1: the most traditional, even though now a little outdated, approach proposed by Reiter and Dale [1997] consisting of grouping similar NLG tasks into modules that are then connected via one-way-pipeline.

In the first place, we would like to highlight the core reasons that led to a chosen approach:

- My personal lack of experience in NLG as this was my first ever NLG system that I have created.
- The lack of easily accessible high amount of data.
- The full extent of an NLG problem: from non-processed data to well-built text.

For me, personally, this field of computational linguistics is new and therefore the aim was not only to build a NLG system, but to gain knowledge including different approaches in this specific subfield of NLP. Personally, staying faithful to the division of modules is the most intuitive and also feasible solution, which closely relates to the point of the end-to-end extent of the problem. It was easy for me to get lost in the high amount of issues to resolve (each of the NLG tasks) without giving it a properly organised structure. The lack of experience also resulted in the error propagation that has risen repeatedly during the development.

In addition, the lack of data almost forbids the data-driven approach. Naturally, the data do exist, but their acquisition would require either a high amount of time-consuming labour of creating the data by hand or acquiring football articles automatically and then aligning them with given input data, which still can end up insufficient as for the acquired text more data could have been known. Such a corpus building could be a project on its own.

5.4 Match example

A match example is provided and the summary can be seen in figure 5.1.

Football match between *Jablonec* and *Bohemians 1905*, which resulted in a 3 to 1 victory for the home team *Jablonec*. First half can be summarised into words to further describe the figure as follows: First goal was a penalty scored by *Trávník* in 10th minute and then second incident was a goal by *Bohemians'* player *Hašek* that tied the game in 44th minute after a pass from *Vaníček*. Events

that happened in the second half are hopefully easy to interpret as well. Numbers next to an incident group them by their type- (1) penalty goal, (2) goal with an assist, (3) substitution and (4) yellow card.

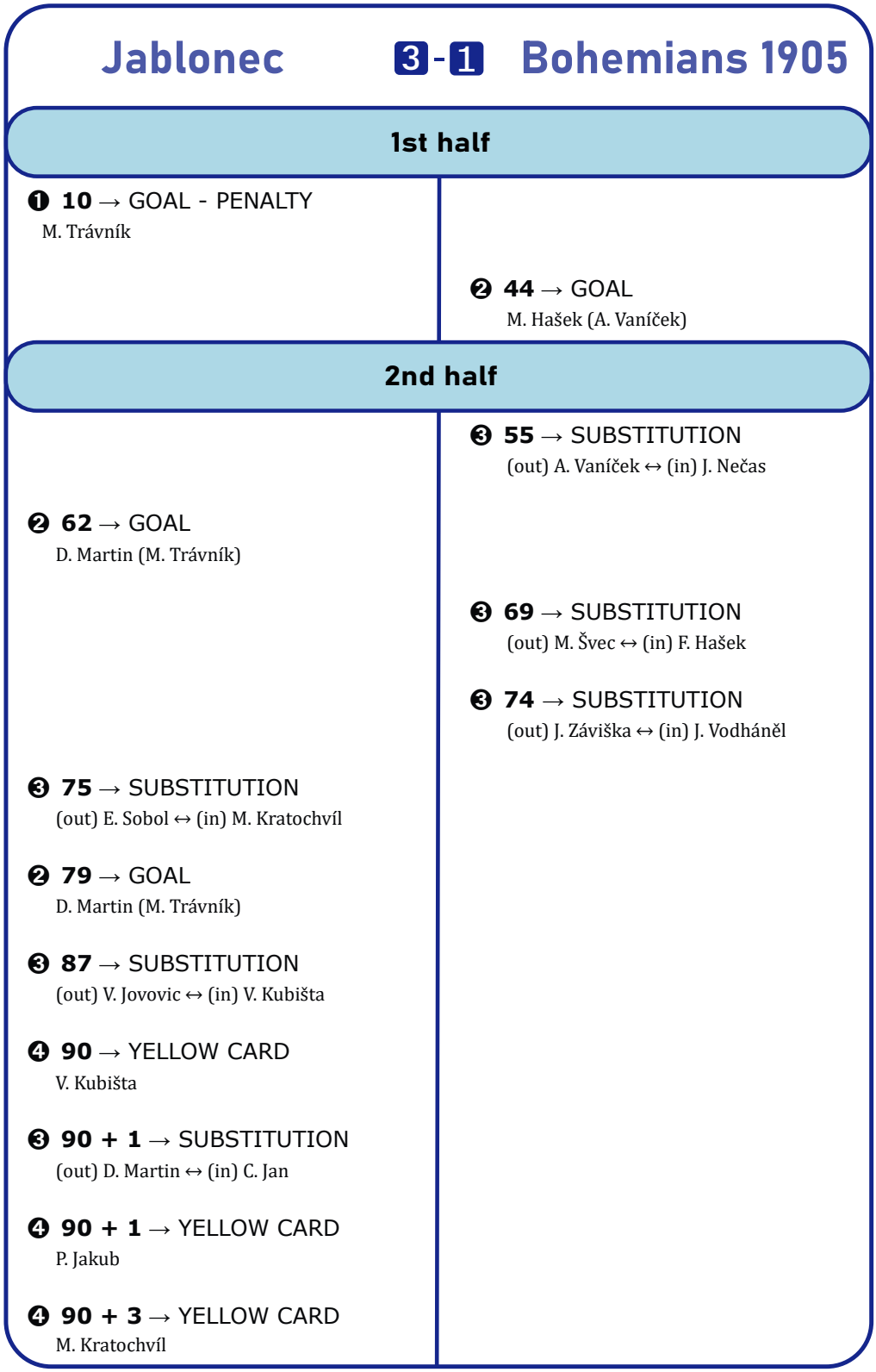


Figure 5.1: Table summary of the example match.

5.5 Structure overview

The overview of the structure of the solution is shown in figure 5.2.

Input data with arguments to modify slight functioning (as a way of output is presented, number of texts to generate, input data or key for Genja) are passed to *run.py* (figure 5.2 - 1) (.py will be left out as it is implicit for every module), which is the executable starting point of the program. Parsed arguments are an input for *articles_generator*, which handles the communication between each module, essentially creating the one-way-pipeline for modules (figure 5.2 - 2, 3, 4, 5) and managing passing correct arguments to functions. Output of this core handler is tuple of strings: title and body of the article.

The *data_initializer* (figure 5.2 - 2) module is not present in the original modular architecture by Reiter and Dale [1997] (described in 3.1). However, since the data are in a raw form, the step of data processing needed to be added to store the data internally in a more convenient way to further operate with the information given effectively.

Modules (figure 5.2 - 3, 4, 5) correspond to the modular architecture perfectly. *Document_planner* (figure 5.2-3) takes data about a match in a convenient inner representation as an input and creates a *DocumentPlan*, which contains ordered preverbal messages to be said in an article. Then these messages are lexicalized in *sentence_planner* (figure 5.2 - 4) and finally linguistically realised by *linguistic_realiser* (figure 5.2 - 5).

Moreover, there are three auxiliary modules (described in detail in 5.6.1) to keep the structure of the modular approach implementation as clean as possible:

- *Data.py*
- *Types.py*
- *printer.py*

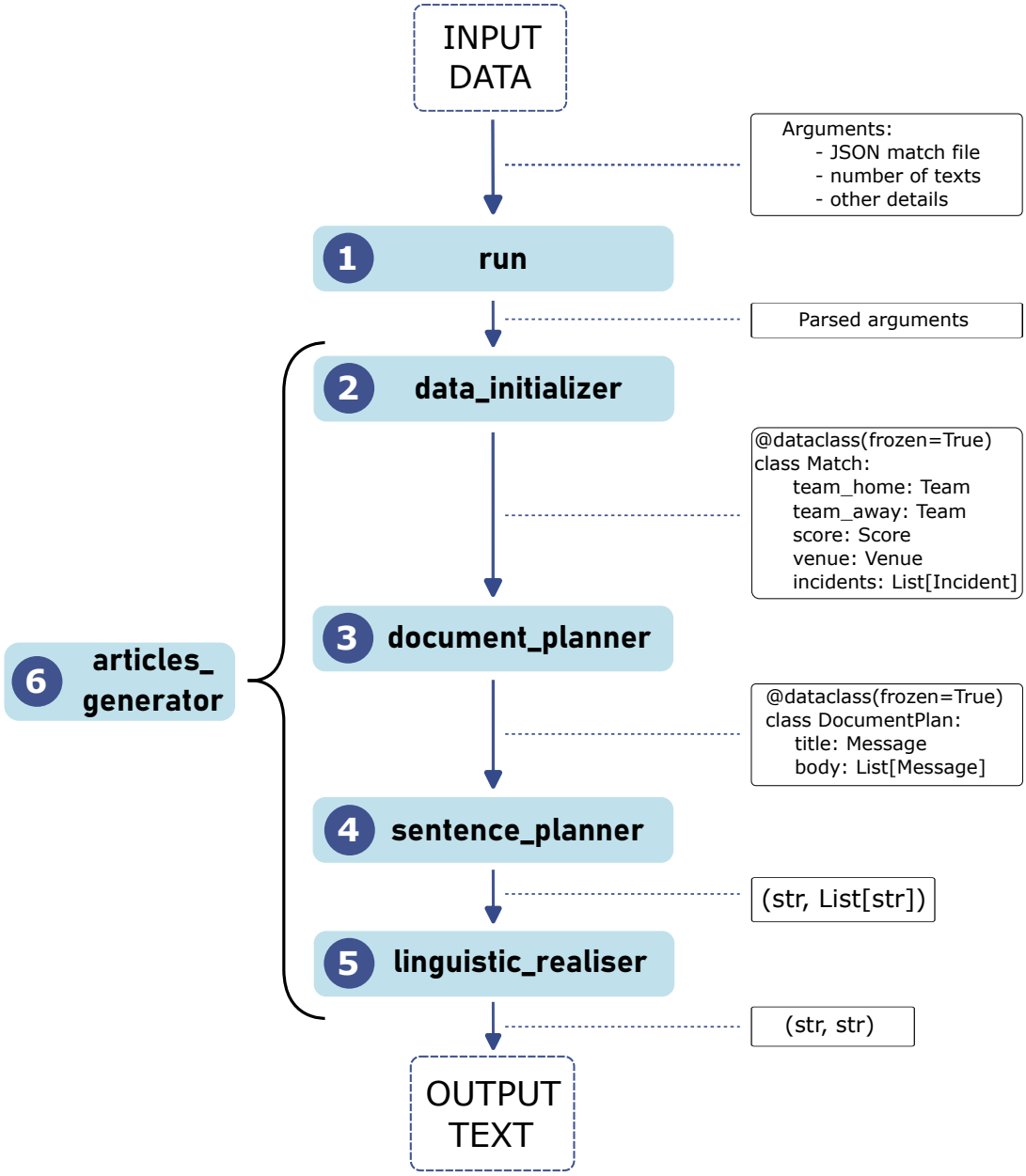


Figure 5.2: Overview of solution structure.

5.6 Modules implementation

In this section the implementation of each module is described in detail creating a compact solution in Python. For even closer inspection, feel free to look into the code.

5.6.1 Auxiliary modules

Data.py groups data classes that store information contained in a JSON file. Namely these entities are *Score, Venue, Country, Player, Team, Time, Incident (+IncidentParent)* and finally the most crucial *Match* that encapsulates each

of the classes mentioned. Note that these classes are made immutable using *@dataclass(frozen=True)* from package *dataclasses* and then implementing static *create* method. Immutability ensures the data remain intact after manipulating with them frequently in multiple functions across modules.

Module ***printer.py*** manages printing different components in a readable way. Since there is a lot of data, even during the development it was more convenient to pretty-print separate results of modules instead of inspecting more nested and complicated states during debugging. Due to numerous alignments and length of string appending, a separate module was created to not overload other segments of the code.

Lastly, ***Types.py*** stores enumerate types for different entities. For instance, in the *document_planner* module there is a class *Message* to store attributes of a preverbal message. One of the attributes is the type of the message defined as *Types.Message*. This class is implemented in *Types.py* (along with other types, which have arisen so often that a separate module was created) as Python's *enumerate()*. Options are *GOAL, PENALTY_KICK_MISSED, CARD, SUBSTITUTION, RESULT*. To give one more example, message that reports a card incident has attribute storing the type of card in *Types.Card - YELLOW, RED_AUTO, RED_INSTANT* (distinguishing the difference between two yellow cards and instant red card - to highlight the severeness of the foul). Similarly, other types of different-level entities are stored.

5.6.2 Data initializer

Goal of the ***Data_initializer*** is to transform data from the initial JSON file to a more convenient inner representation implemented as a system of classes to uphold the principles of object-oriented programming resulting in a crisp and well-divided structure.

The implementation is straightforward - we use the built-in package *json* to ease the manipulation with a JSON file while initializing classes from the ***Data*** module.

Two examples of specific records contained in the JSON file and their Python representation are shown in figure 5.3 and figure 5.4.

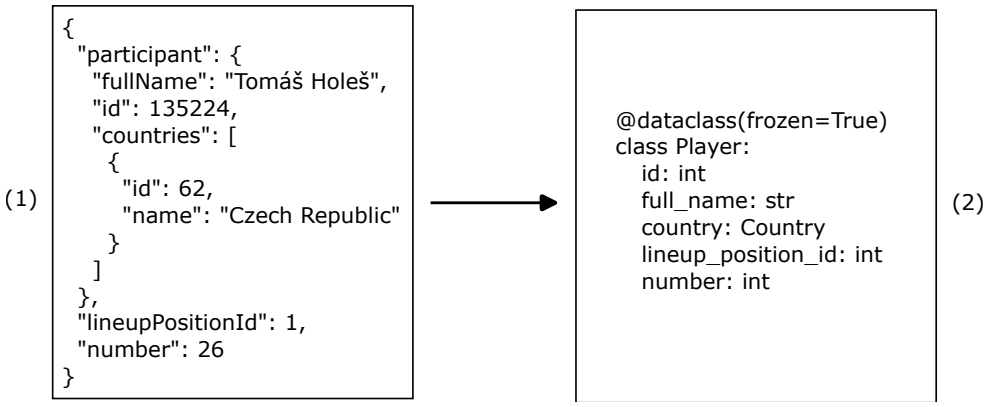
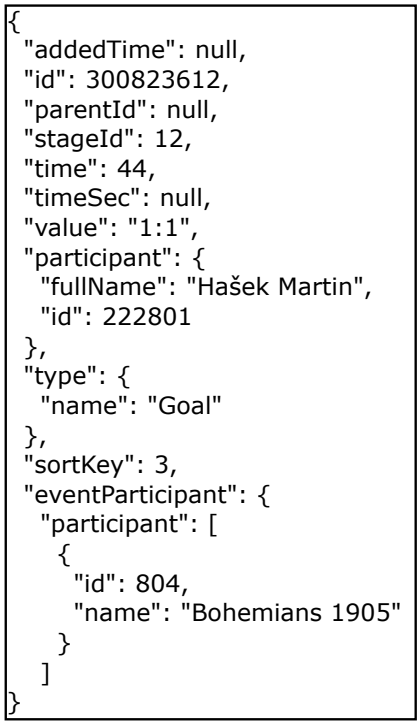


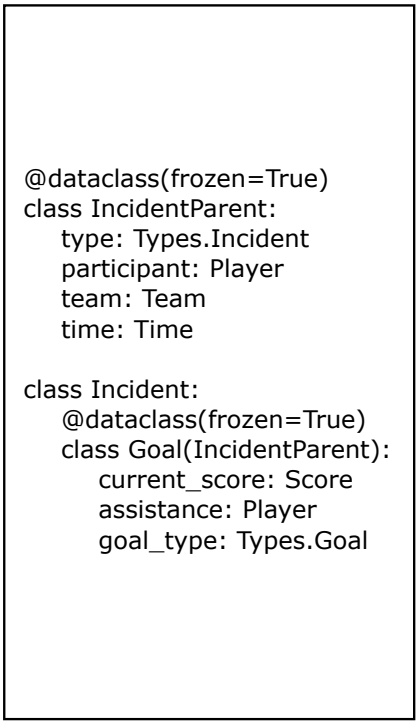
Figure 5.3: Transformation of entity *Player* from JSON to a Python class.

Entity Player is transformed from its initial JSON representation (figure 5.3

- 1) to a easy-to-work-with Python's immutable class *Player* with corresponding attributes. Note that a country has also a proper class *Data.Country*.

(3)





(4)

Figure 5.4: Transformation of entity *Incident* from JSON to a Python class.

Entity *Incident* is transformed from JSON (figure 5.3 - 3) to a representation as an immutable Python's class (figure 5.3 - 4). The class is called *Goal* and it is a subclass of *Incident*. This particular class structure ensures that different types of incidents (goal/card/substitution/penalty kick) can be addressed as one variable type and therefore enabling the *Data.Match* class to have attribute *incidents* storing incidents as *List[Incident]*. Furthermore, every *Incident* subclass also inherits from *IncidentParent* class since these are common attributes for every match incident and it would be inefficient to repeat those attributes multiple times.

Note that this module resolves part of content determination since the number of attributes from the initial JSON file are not transformed into Python class. For instance, starting time of the match was not saved since we mark this information as redundant for an article generating. Similarly, more attributes were omitted to avoid useless information. On the other hand, couple attributes end up redundant (e.g. *Country* of *Player*), but these attributes are kept to ensure integrity of the solution and enable an easier further future development.

5.6.3 Document planner

The aim of this module is to plan the content of the separate messages and their order. Document planner is resolved trivially - each incident is transformed to a separate preverbal message along with fundamental data.

Implementation is shown in figure 5.5 and is similar to *Incident* structure. Each preverbal message has different arguments and therefore its own class. To access *messages* classes as one variable type there is a system of inheritance, where every message class inherits from *MessageParent* so that they can be distinguished by their according type *Types.Message*. Note that these classes are made again immutable to ensure data stability.

Unlike message class *Substitution*, which is self-explanatory, classes *Result* and *MissedPenalty* need clarification.

Result message defines the title of the article. The core purpose of the title is to summarise a match in one sentence and hence to report the result/score of the match.

Content of the *MissedPenalty* is rather obvious, but the reasoning behind the existence of this separate class may not be visible at first glance and relies heavily on domain knowledge. Penalty is a crucial, rare and thrilling event during a football match.[1] Consequently, every penalty is worth mentioning regardless of the outcome. However, if the penalty is successful, the outcome is a goal and we would like to treat the message the same way as any other goal. I believe every goal in a football match should be reported since scoring in football is rarer than in other sports in general (compared e.g. to basketball where reporting every basket would be too specific and unnecessary). Contrastingly, the message that reports a missed penalty is not strictly required to be present in the article as this event did not affect the result directly. On the other hand, such a message creates rather a shocking element of the article as players are expected to score from this position. This paragraph further illustrates the hand-crafted principles based on the domain knowledge that are incorporated in the solution.

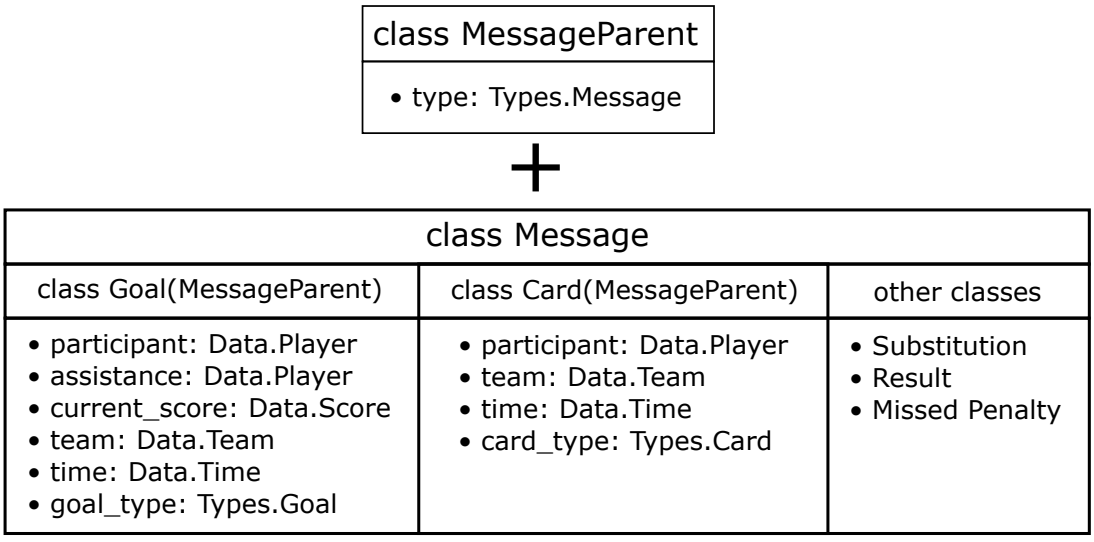


Figure 5.5: Implementation of preverbal messages.

[1]Remember the famous penalty by Antonín Panenka in European Championship finals in 1976 that led to triumph of Czechoslovakia. The kick was groundbreaking and nowadays the term 'Panenka' refers to a style of kick, where you only chip the ball in the middle of the net.

5.6.4 Sentence planner

Last but not least is a *sentence planner* module, that combines sentence aggregation, lexicalization and REG. Due to a possible low number of messages to convey, the result depends heavily on the course of the game and we do not want to leave out any of the incidents that occurred during the match. So to further simplify the process, one incident corresponds to one sentence and the sentence aggregation is not performed. Sentence aggregation as described in 2.3, is grammatically a non-trivial operation and we leave this part as a room for improvement.

The core idea is to exploit templates, which means inserting words to express a constituent (e.g. entities to be expressed from non-verbal message) verbally in the reserved empty spot in the sentence. However, to ensure some kind of variability of the expressions as well as easy cooperating with Genja API the templates are resolved in a bit more complex way. Naming classes and variables throughout *sentence planner* was difficult due to the similarities of the segments that needed to be named, therefore I first present crucial classes and reveal their overall integration later.

Firstly, the template, by its very nature, is represented as a class called ***Sentence***. *Sentence* controls the structure of the sentence (ordering and selection of constituents) and allows other components of the implementation to insert various lexical items. *Sentence* contains string *id* to simplify the number of types and subtypes in this section. Boolean *simple* says whether the expression is simple, clear, syntactically easy or the opposite: colourful, longer and more complex. Then there is a list of *constituents*, which is either a hand-inserted string or *Constituent*, that requires further development and will be picked later. Note that *Sentence* contains an attribute *msg*, but the attribute is not assigned immediately.

```
class Sentence:
        id: str
        simple: bool
        constituents: List[Union[str, Constituent]]
        msg: dp.Message
```

Here is an example of one *Sentence* initialization, which defines the easiest possibility of how to express a message that a goal was scored. Furthermore, the following table shows possible lexicalization of these constituents in English(EN) and Czech(CZ) to illustrate the principle of such a structure.[2]

```
Sentence.create(type_, subtype, True, [
  Constituent(id_='e-time', morph_params='',
          explicit_data=Types.ExplicitEntityData.TIME),
  Constituent(id_='v-goal', morph_params='.-0-.-.-.',
          explicit_data=None),
  Constituent(id_='e-player', morph_params='1-.-0-.-.',
          explicit_data=Types.ExplicitEntityData.PARTICIPANT),
  Constituent(id_='w-goal', morph_params='4-.-.-.-.',
          explicit_data=None)]))
```

	Time	Verb for goal	Player entity	Word for goal
	e-time	v-goal	e-player	w-goal
EN	In 7th minute	scored	Ronaldo	goal.
CZ	V 7. minutě	dal	Ronaldo	gól.

Another class is ***Constituent*** containing again *id* to mark its type or subtype. In Czech multiple agreements as well as grammatical cases need to be resolved and therefore we have to store such information when creating sentence layout regardless of the lexical expression that will be inserted. All morphology attributes are stored in *morph_params* as containing case, tense, gender and ref/agr all to be then handed over to Genja API for linguistic realisation. Again, as seen in the *Sentence* initialization, *morph_params* are initialized shortly using string *id*.

```
class Constituent:
        id: str
        morph_params: MorphParams
        explicit_data: Types.ExplicitEntityData
        string: str
```

Lastly, class ***Template*** handles the lexicalization of every constituent.

```
class Template:
        id: str
        string: str
```

Templates can be strings, or a part of entity information can be inserted into a lexical item. *Template*'s string can be defined by hand, or by function or by creating a template (by combining string and variables' string values to be inserted), which expresses the constituent. Here are examples for templates that express player entity and a word for assistance.

[2]Note that the word order is not correct in the English sentence, but correct in the Czech one. My translation will mainly illustrate the process for English readers, which can lead to grammatically incorrect sentences or expressions.

Entity player (e-player)		
String template	EN	CZ
player.full_name	Cristiano Ronaldo	Cristiano Ronaldo
player.get_last_name()	Ronaldo	Ronaldo
f"hráč číslo {player.number}"	player number 7	hráč číslo 7

Word assistance (w-assistance)	
EN	CZ
assistance	asistence
pass	přihrávka

To recapitulate, class *Sentence* creates templates for sentence combining and ordering numerous constituents, which are either hand-written string or represented as an internal *Constituent* class. For every *Constituent* an instance of the class *Template* is created to then present options for expressing *Constituent* in numerous ways. Note that to store a string, data to insert need to be assigned when initializing *Template* (unlike *Sentence*, where *Message* can be added later).

To manage these objects and enable their cooperation, two handlers are created: *SentenceHandler* and *TemplateHandler*. *Sentences* are first initialized all, then randomly shuffled and lastly grouped by their id and always using a simple sentence first. Then for each message in the article a *Sentence* is picked in the order they were grouped and message is assigned to the *Sentence* as attribute *msg*. Once every *Sentence* for a given message type is used, we pick randomly from used. This process ensures structural variability as we use the same *Sentence* twice only in case that we used all of the *Sentences* for a given message.

To further increase variability of the created sentences a similar process is repeated for *Templates*. *Templates* are randomly picked from non-used options. Once every template is present in the output, we pick random from every template ensuring we do not use the same template twice in a row. Another difference is that *Templates* are first initialized all with empty values (where string is expected to be inserted) and then *Template Handler* frequencies are updated to set up the initial state of *Templates*.

5.6.5 Linguistic realiser

Czech language is quite complex in terms of grammatical difficulty and therefore resolving the problem of realising chosen lexical items correctly by-hand without a proper tool performing complex morphology transformations is not efficient. This is the last step of the entire NLG process, however, without the realisation module the outputted text will not be readable. Consequently, other software was incorporated into our solution in order to perform linguistic realisation.

The system is called Genja and was provided by Geneea(see `https://geneea.com/`). The Genja system, which was based on Jinja, manages working with smart templates. In order to exploit templates efficiently, Genja offers tools for managing morphology and knowledge base. Note that authorization key is required in order to perform linguistic realisation. In the solution's default, the is key stored as a system environment variable to not be visible in a public GitHub repository. Without the key, the program will still run, but the linguistic

realisation will not be performed and therefore the outputted text will be the intended input for Genja.

Due to the strict syntax that needs to be inputted in a Genja API request, a number of transformations need to be done. Firstly, combining plain strings in a template with morphological parameters creates a well-built input for Genja. Secondly, a json file for Genja must be created and lastly the output must be transformed from json back to string. After that the auxiliary file is deleted.

5.6.6 Articles generator

To finalise, articles generator connects modules together creating a notional one-way pipeline.

5.7 Output

Since the lexical items and sentences' word orders are picked randomly, the output can contain a combination of expressions that are suboptimal and hard-to-read. For example when describing substitution message, realisation of two player entities can result in unappealing text, where it is not clearly visible who is the first and second player (player Vodháněl was subbed for player Kratochvíl, whose name was lexicalized as last name and first name):

V 74. minutě střídal Vodháněl Kratochvíla Miloše.(CZ)

In 74th minute subbed Vodháněl Kratochvíl Miloš.(EN)

Moreover, due to the international nature of the football domain, often finding grammatically correct forms of foreign names is not easy and Genja can not resolve this issue.

To compensate for these minor problems we output multiple texts instead of just one. As said, randomness can highly affect the overall impression from the text. When generating multiple texts the probability that one output will feel better than others is then increased. The idea of generating more than one text and then picking the best result of all generated can be useful, when the quality of the outputted text can not be ensured. However, the process of selecting "the best" result can be again approached differently. The easiest solution is ,for instance, using a human evaluation. Another option is to exploit the data-driven methods and pick the version that is statistically the most similar to the texts in the acquired set of data e.g. football articles from websites. The problem of selecting "the best" option is not solved in this project.

Here are multiple examples of lexicalization and realisation of one preverbal message containing information about player being penalized:

- Message: (Type: CARD, time: 90 + 1, participant: Povazanec Jakub, team: Jablonec, card_type: YELLOW)
- *(EN) In the first minute of the overtime Jakub Povazanec got a yellow card.*
- *(CZ) V první minutě nastavení druhého poločasu obdržel hráč s číslem 7 žlutou.*

- *(CZ) Jakub Povazanec vyfasoval jednu minutu po začátku nastaveného času druhého poločasu žlutou.*
- *(CZ) Povazanec dostal jednu minutu po začátku nastaveného času druhého poločasu žlutou kartu.*

Note that both word order and lexical expressions for the constituents differ: player (Jakub Povazanec/Povazanec/hráč s číslem 7), yellow card (žlutou/žlutou kartu) or the verb for "got"(obdržel/vyfasoval/dostal). As you can see the template system used created a variety of well-built sentences.

Now we show two examples of generated articles for the *example_match*. Messages to generate are visible from figure 5.1, but we will state couple of first along with the title message:

Messages:

1. Type: RESULT, team_home: Jablonec, team_away: –Team– Id: 804, Name: Bohemians 1905, type: AWAY, score: 3:1
2. Type: GOAL, time: 10, participant: Trávník Michal, team: Jablonec, score: 0-0, goal_type: Goal.PENALTY
3. Type: GOAL, time: 44, participant: Hašek Martin, team: Bohemians 1905, score: 1-1, goal_type: Goal.ASSISTANCE
4. Type: SUBSTITUTION, time: 55, participant_out: Vaníček Antonín, participant_in: Nečas Jakub, team: Bohemians 1905
5. ...

Article examle No. 1/2:

Jablonec deklasoval Bohemians 1905 3:1.

V desáté minutě proměnil Trávník pokutový kop. Hráč s číslem 21 vstřelil po nahrávce Vaníčka Antonína 44 minut po začátku gól. V 55. minutě Jakub Nečas střídal za hráče s číslem 22. 62 Minut po začátku dal po nádherné kombinaci Doležal po pase Trávník Michal branku. V 69. minutě vystřídal Hašek za Michala Ševce. Hráč s číslem 17 střídal za Jana Závišku 74 minut po začátku. V 75. minutě Kratochvíl Miloš vystřídal hráče s číslem 21. Doležal Martin vsítil po asistenci Trávník gól 79 minut po začátku druhého poločasu. 87 Minut po začátku střídal Vojtěch Kubišta hráče s číslem 25. Kubišta dostal v 90. minutě žlutou kartu. Jednu minutu po začátku nastaveného času druhého poločasu vystřídal hráč s číslem 19 Doležala Martina. V první minutě nastavení druhého poločasu obdržel Povazanec žlutou. Kratochvíl Miloš vyfasoval tři minuty po začátku nastaveného času druhého poločasu žlutou kartu.

Article example No. 2/2:

Jablonec porazil Bohemians 1905 3:1.

V desáté minutě proměnil Michal Trávník penaltu. Hašek dal po přihrávce Vaníčka Antonína 44 minut po začátku branku. 55 Minut po začátku druhého poločasu vystřídal hráč s číslem 10 Vaníčka. Po nádherné souhře v 62. minutě vstřelil Martin Doležal po asistenci Trávník Michal gól. 69 Minut po začátku druhého poločasu Hašek střídal za hráče s číslem 2. V 74. minutě vystřídal Jan Vodháněl za hráče s číslem 8. Miloš Kratochvíl střídal za Sobola Eduarda 75 minut po začátku. 79 Minut po začátku druhého poločasu vsítil Martin Doležal po pase Trávník branku. 87 Minut po začátku Kubišta Vojtěch vystřídal Vladimír Jovovič. Devadesát minut po začátku druhého poločasu obdržel Kubišta Vojtěch žlutou. V první minutě nastavení druhého poločasu střídal Chramosta za Martina Doležala. Povazanec dostal jednu minutu po začátku nastaveného času druhého poločasu žlutou kartu. V třetí minutě nastavení druhého poločasu vyfasoval Kratochvíl Miloš žlutou.

Football articles generated by the software are readable, fluid and somewhat variable. Outputted texts are with only a handful of mistakes (grammatical, not factual). In the 4th sentence of the first article, *Trávník Michal* has an incorrect form and Genja did now manage to process this: the correct form for the 2th case is *Trávníka Michala*. Same problem occurred in the second article for the name *Vladimír Jovovič*. Also in the last sentence of the second article, *V třetí* is not grammatically and the form should be *Ve třetí* instead.

5.8 Discussion

In this section we discuss numerous perspectives and aspects of the solution.

Several papers on the topic of football were published. First example is et Theune [1997] where a system of templates is incorporated in order to generate a football report. Syntactic trees are used to implement templates. Another difference is the ordering of the messages. While our implementation orders messages chronologically, this paper does the following: first we choose the topic we have not spoken about yet e.g goals scored during the match, then we convey every message from this topic. However, the ordering of topics can change. Consequently, we need a tool that controls the state of the generation so that we know what was or was not already mentioned (e.g. number of spectators will be mentioned in a game summary only if it was not mentioned in another topic). Context supervision is present throughout the entire generation.

Second example of the implemented NLG system about football is much newer Chen and Mooney [2008]. This system generates commentaries to events on the pitch in the simulated football match. Their approach is completely non-linguistic and relies on machine learning methods. Training data consists of pairs of textual human commentary and state of the game. These two examples show two completely different approaches.

Next to discuss is the choice of the approach used in this implementation. The architecture strictly follows the division described in a modular approach exploiting the power of splitting the extremely complex NLG problem into smaller easier-to-manage segments. As discussed earlier, the raw power of data can contribute to the quality of the result and therefore the first improvement of the project would be to change the approach completely assuming you acquire the

corpus data. However, a stochastic approach relies on a big amount of quality data, which are not available for me or rather their requirement would be insanely time-consuming. Moreover, this was my first NLG project. Due to my personal inexperience and lack of a large number of easily accessible data the modular approach was chosen.

Naturally, the quality of football articles created by sport journalists can hardly be matched with the quality of outputs from the *FootballArticlesGenerator* software. The aim of the outputs were to be readable and form a proper Czech sentence filled with information about a football match, which was achieved. What is more, an increased variability was achieved to make text more fluid.

The complexity of the implementation was hard to deal with and not seeing the error propagation consequences the attempts of creating a fully functioning system often failed. Retrospectively, the incorporated system of "two-layered" templates (implemented as *Sentence* and *Template*, which both fit to the template idea) is not-optimal. As seen in initializing *Templates* for *Time*. Firstly, the templates are picked randomly without context, meaning we can use expressions like "Two minutes after that" and so on. Secondly, to express time more variably we would like to use another template layer connected to a set of morphological parameters to be passed to Genja, but that is not possible. The best improvement for this project would be to re-implement sentence planner and using tree structure of templates to enable having template in a template.

Such a tree structure would then require some non-trivial design and implementation to deliver the expected result. Sentences represented as trees where nodes are either strings or templates should be pre-created to ensure variability between the sentences used throughout the text. Then the sentence aggregation can be performed once the content of the sentences is known. Lastly, lexicalizing each template in the sentence tree by various expressions. The core problem when designing this system is interconnection between the tasks, that requires a decent amount of flexibility of the tree structure in terms of implementation. For instance, sentence aggregation can merge two of the trees together.

Consequence of the well-implemented tree structure would be a better quality article. Due to the kept approach of using templates, the level of the result would not still be near to texts produced by humans. Templates offer only a number of varieties in comparison to a complex human language. Naturally, the higher the number of templates the more variable the text will be, initially approaching the human-like level.

In addition, the database used for this project does not contain enough information to create a somewhat complex article. Primary insufficiency is in the lack of details of incidents contained in the data. Expressing a goal is usually accompanied by a slight description of how the goal was scored: cross from the left side and top corner header, chaos in front of the net resulted in a rebound and a goal, beautiful solo play, 30 metre absolute screamer, etc. Similarly, the minor incidents like corners, shots, fouls, offsides are not present and could help when conveying more detailed text describing the true course of the game and not only major incidents. Lastly, there are no statistics and knowledge base present. When lexicalizing Cristiano Ronaldo we have only options to use number or name, while journalist would use more complicated expressions like "deadly portuguese striker", "famous CR7", "the most productive player of the Premier League or

"the newest member of Manchester United squad". History of the player, current leaderboards and statistics of the season, all time best leaderboards, nicknames for entities and more are the information lacking to truly have potential to create human-like articles that would be both thrilling and informative. Not to mention incidents, which are not based on statistics, but rather on the actions that happened on the pitch. For instance, controversial calls, conflicts between players, vibes on the stadium, how well the team actually plays are all aspects that are very hard to store, classify and express, but are arguably the most interesting for a football fan besides the result of the match itself. This paragraph further proves the strength of the data-driven approach that can overcome the difficulties of lack of data by simply imitating the already existing text. For example, if a corpus contains the nickname "Blues" for "FC Chelsea" numerous times, it becomes a viable lexical expression for the FC Chelsea entity.

Even though the generated texts are well-built and correct, their usage afterwards is questionable. The popularity of sports articles lie in the tabloid style nature of the text. If they are interested in just a simple overview of the match, they use non-textual representation like figure 5.1, which is provided online (e.g. Livesport). Moreover, the input for the program is very specific and requires an authorization key for Genja.

Personally, I am really happy with the result. The NLG system I have created is end-to-end and transforms raw data into a well-built text, which is non-trivial and variable to some extent. However, the core consequence I am glad for is the amount of knowledge I have gained in the field of NLG. I hope that somebody will use this text as a learning material when exploring the field of NLG and will find the information useful. I truly recommend this text for someone, who is preparing himself to develop his first NLG system, as the thesis often contains passages, where I struggled with the explanation of why.

Conclusion

The aim of this thesis was to implement an NLG system that generates football articles from non-linguistic data. Along with the implementation, a brief NLG overview was required in order to have baseline knowledge for creating the intended software. We described the process of developing an NLG system end-to-end with tasks that are needed to take into consideration in order to create a fully functioning solution. We have covered the basics of NLG such as different approaches and their core benefits and drawbacks. Moreover, we have illustrated numerous problems that can arise during the development and that are required to be dealt with beforehand.

In the second section we present an implementation of such a text-generating system that uses slightly outdated (for reasons stated throughout the thesis multiple times), but, on the other hand, accessible methods. The implementation is described in detail. Choices made across the solution are usually explained as well. To inspect the implementation even more, please look into the code itself.

As stated in the end of the previous section, this work may not have that many implications. Although software can be used to present the course of the match in a textual and less-schematic representation, the quality of the outputted text is not on the newspaper level (which was not expected). Another problem with incorporating this project into other systems is the specificity of the input and the usage of linguistic realiser Genja, which is not available to the public. However, pieces in between said segments can be used. What is more, this paper can be utilised by other computer scientists, who lack complex insight into the field of computational linguistics and would like to study the subfield of NLG, since the overview of NLG is written from the perspective of a beginner. Furthermore, the text is filled with easy-to-follow examples and possible obstacles during the development to further illustrate knowledge carried across the paper.

Bibliography

Regina Barzilay and Mirella Lapata. Aggregation via set partitioning for natural language generation. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 359–366. Citeseer, 2006.

Giuseppe Carenini and Johanna D Moore. Generating and evaluating evaluative arguments. *Artificial Intelligence*, 170(11):925–952, 2006.

David L Chen and Raymond J Mooney. Learning to sportscast: a test of grounded language acquisition. In *Proceedings of the 25th international conference on Machine learning*, pages 128–135, 2008.

Aggeliki Dimitromanolaki and Ion Androutsopoulos. Learning to order facts for discourse planning in natural language generation. *arXiv preprint cs/0306062*, 2003.

Pablo A Duboue and Kathleen McKeown. Statistical acquisition of content selection rules for natural language generation. 2003.

Mari et Theune. Generation of soccer reports: some results. 1997.

Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

Albert Gatt and Emiel Krahmer. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research*, 61:65–170, 2018.

GPT-3. A robot wrote this entire article. are you scared yet, human? *The Guardian*. URL `https://www.theguardian.com/commentisfree/2020/sep/08/robot-wrote-this-article-gpt-3`.

Mika Hämäläinen et al. Harnessing nlg to create finnish poetry automatically. In *Proceedings of the ninth international conference on computational creativity*. Association for Computational Creativity (ACC), 2018.

Steven Johnson and Nikita Iziev. A.i. is mastering language. should we trust what it says? *The New York Times*. URL `https://www.nytimes.com/2022/04/15/magazine/ai-language.html`.

Ioannis Konstas and Mirella Lapata. A global model for concept-to-text generation. *Journal of Artificial Intelligence Research*, 48:305–346, 2013.

Chunyi Liu, Peng Wang, Jiang Xu, Zang Li, and Jieping Ye. Automatic dialogue summary generation for customer service. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1957–1965, 2019.

Tomasz Marciniak and Michael Strube. Beyond the pipeline: Discrete optimization in nlp. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 136–143, 2005.

David D McDonald. Issues in the choice of a source for natural language generation. *Computational Linguistics*, 19(1):191–197, 1993.

David D McDonald. Natural language generation. *Handbook of Natural Language Processing*, 2:121–144, 2010.

Ratish Puduppully, Yao Fu, and Mirella Lapata. Data-to-text generation with variational sequential planning. *arXiv preprint arXiv:2202.13756*, 2022.

Adwait Ratnaparkhi. Trainable methods for surface natural language generation. *arXiv preprint cs/0006028*, 2000.

Ehud Reiter and Robert Dale. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87, 1997.

Graeme Ritchie. Computational mechanisms for pun generation. In *Proceedings of the Tenth European Workshop on Natural Language Generation (ENLG-05)*, 2005.

Somayajulu Sripada, Neil Burnett, Ross Turner, John Mastin, and Dave Evans. A case study: Nlg meeting weather industry demand for quality and quantity of textual weather forecasts. In *Proceedings of the 8th International Natural Language Generation Conference (INLG)*, pages 1–5, 2014.

Kavita Thomas and Gowri Somayajulu Sripada. Atlas. txt: Linking georeferenced data to text for nlg. 2007.

Xianhua Zeng, Li Wen, Yang Xu, and Conghui Ji. Generating diagnostic report for medical image by high-middle-level visual information incorporation on double deep learning models. *Computer Methods and Programs in Biomedicine*, 197:105700, 2020.

List of Figures