Foundations for reusable and maintainable surface realisers for isiXhosa and isiZulu

Zola Mahlaza

A thesis presented for the degree of

Doctor of Philosophy

in the Department of Computer Science

University of Cape Town

August 2022

University of Cape Town

The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

University of Cape Town

For Nokulunga Mahlaza and Cebisa Xhamane.

Acknowledgements

Baninzi gqithi abantu endifuna ukubabulela ngokundicendisa kwizifundo zam. Andinokwazi ukubabhala bonke apha nokuba bendifuna.

Ndiyabulela kwi Hasso Plattner Institute (HPI) ngemali yokufunda kunye nokundisa emazweni.

Ndibulela kakhulu kuSomila Fuma nakuSophakama Mahlaza ngondixhasa kunye nokundiboleka indlebe.

Bendingasoze ndifike apha ukuba zange ndifumane inxaso ngoku ndandisekwisikolo samabanga aphezulu. Ndifuna ukuthi ndibamba ngazo zozibini kuSiphosethu Mnene, Sive Bukula, Lutho Bongco, Ukho Dondashe, Avukonke Jona, Odwa Ntsephe, Zolani Nombembe, Vuyisa Maswana, Amangwe Vatsha, Lindokuhle Vatsha, Sibusiso Mqhayi, Nangamso Matshanda, Aphiwe Ntsini, Siphosethu Martins, Sithembele Dick, Aphiwe Veveza, Zintle Mazaleni, kunye nabanye abaninzi gqithi. Enkosi kubo bonke oo-Tishala base Hector Petersen, kwaZwelitsha.

Ndiyabulela kumntu wonke ebeyingxenye yamaqela eDigital Libraries, KnowledgE ENgineering, kunye neHuman Computer Interaction kwiDjunivesithi yaseKapa. Enkosi kuLighton Phiri, Joan Byamugisha, Richard Maliwatu, Toky Hajatiana Raboanary, Hafeni Mthoko, Jecton Tocho Anyango, Joseph Telemala, Tezira Wanyana, Wanjiru Mburu, Sarah Dsane, Frances Gillis-Webber, William Leighton Dawson, Mary-Jane Antia, Jackson Moji, Mustafa Ali, ukuya kwabanye abaninzi gqithi.

Last but not least, I am tremendously grateful to my supervisor, Assoc. Prof. C. Maria Keet. I was able to grow intellectually because of her tough questions, her frankness, and the wonderful environment she created for the KnowledgE ENgineering team. Her guidance, direct and indirect, since 2014, allowed me to break down a wall I have kept between scientific endeavours and "real life" for most of my life. A wall I've kept because of incorrect, and dare I say idiotic, conceptions on who gets to be a scientist and what kind of problems scientists (or "true scientists") ought to work on. Without that, I would have never been able to appreciate the impact I could make in this world. Heel hartelijk bedankt, Maria.

Abstract

Natural Language Generation (NLG) systems are used to generate text in order to reduce manual effort. Most existing systems are built to support European languages with simple and/or well-documented grammars. IsiZulu and isiXhosa, two of the largest South African languages by first language speakers, have not received a lot of attention in the field despite the potential impact of NLG systems for their speakers. The existing NLG systems created for these languages rely on *ad hoc* methods for surface realisation. Surface realisation is the process of generating text from a system's abstract representations of sentences. The aforementioned methods combine templates and grammar rules since the languages are low-resourced and grammatically rich. However, do not use their scant linguistic resources efficiently, they do not rely on a template specification that supports interoperability, and do not use an architecture that yields easy-to-maintain software since none exists.

The objectives of this thesis are to create the foundations for easy to maintain and reusable surface realisation tools for isiXhosa and isiZulu by establishing a principled way to pair templates and grammar rules, organise surface realisation modules such that the components are modular, analysable, and reusable, and create template specifications that are interoperable. In addition, it is to demonstrate that aforementioned objectives can be achieved while generating good quality isiXhosa and isiZulu text in the data-to-text and knowledge-to-text areas.

We achieve these objectives by developing a model-based approach of pairing templates and Computational Grammar Rules (CGRs) to obtain linguistically well-founded templates that are suitable for low-resourced and grammatically rich languages. To obtain interoperable template specifications, we created a task ontology using a bottom-up approach and evaluated it via the standard practice of using Competency Questions (CQs) and removing inconsistencies via an automated reasoner. We also created an architecture that satisfies the most maintainability features from the BS ISO/IEC 25010:2011 standard. In addition, we created proof-of-concept text generation tools that use the proposed approaches and artifacts to generate isiZulu and isiXhosa text and surveyed speakers of the two languages to establish the quality of the text. We have found that most (57%) of the generated isiXhosa texts are judged positively and there is no consensus on the remaining texts, possibly due to differences in dialect. In addition, most (83%) of the generated isiZulu texts are also judged positively as they have at most one participant who considers them to be ungrammatical and unacceptable.

Declaration of Authorship

I, Zola Mahlaza, hereby declare that the work on which this thesis is based is my original work (except where acknowledgements indicate otherwise) and that neither the whole work nor any part of it has been, is being, or is to be submitted for another degree in this or any other university. I authorise the University to reproduce for the purpose of research either the whole or any portion of the contents in any manner whatsoever.

Signature:

Date: October 17, 2022

Publications

Versions of the research presented in this thesis have previously appeared in the following publications:

- **Z. Mahlaza** and C. Maria Keet. A classification of grammar-infused templates for ontology and model verbalisation. In: Emmanouel Garoufallou, Francesca Fallucchi, and Ernesto William De Luca, editors, Metadata and Semantic Research - 13th International Conference, MTSR 2019, Rome, Italy, October 28-31, 2019, Revised Selected Papers, volume 1057 of Communications in Computer and Information Science, pages 64–76. Springer, 2019
- **Z. Mahlaza** and C. M. Keet. "Formalisation and classification of grammar and template-mediated techniques to model and ontology verbalisation". In: International Journal of Metadata, Semantics and Ontologies 14.3 (2020), pp. 249-262
- **Z. Mahlaza** and C. M. Keet. "OWLSIZ: An isiZulu CNL for structured knowledge validation". In: Proceedings of 3rd International Workshop on Natural Language Generation and the Semantic Web, WebNLG+ 2020, Dublin, Ireland (Virtual), December 18, 2020.
- **Z. Mahlaza** and C. M. Keet. "ToCT: A task ontology to manage complex templates". In: Emilio M. Sanfilippo et al., editors, Proceedings of the Joint Ontology Workshops: FOIS Ontology showcase track, co-located with the Bolzano Summer of Knowledge, BOSK 2021, Virtual & Bozen-Bolzano, Italy, September 11-18, 2021, volume 2969 of CEUR Workshop Proceedings. CEUR-WS.org, 2021
- **Z. Mahlaza** and C. M. Keet. "Surface Realisation Architecture for Low-Resourced African Languages". In: ACM Trans. Asian Low-Resour. Lang. Inf. Process. (Oct. 2022). `https://doi.org/10.1145/3567594`. (in print)

Auxiliary publications

Research associated with this thesis, but not explicitly included here, has appeared in the following publications:

- **Zola Mahlaza**, C. Maria Keet, Jarryd Dunn, Matthew Poulter (2021). An evaluation of template and ML-based generation of user-readable text from a knowledge graph. CoRR, abs/2106.14613.
- C. Maria Keet, **Z. Mahlaza**, and M.-J. Antia. CLaRO: A controlled language for authoring competency questions. In Emmanouel Garoufallou, Francesca Fallucchi, and Ernesto William De Luca, editors, Metadata and Semantic Research - 13th International Conference, MTSR 2019, Rome, Italy, October 28-31, 2019, Revised Selected Papers, volume 1057 of Communications in Computer and Information Science, pages 3–15. Springer, 2019

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Several institutions employ humans to analyse data and extract insights using their domain knowledge, and then ask them to compile textual reports detailing their interpretations for different audiences. For instance, numerical weather prediction data is interpreted by weather forecasters in institutions like the South African Weather Service (SAWS)[1] and then the forecasters prepare and disseminate text reports to the general public to keep them informed. Similarly, financial institutions produce reports and slide decks to present their analysis of financial data to their clients. A big bottleneck in such processes, especially for financially constrained organisations, is that manual document preparation is not scalable. For instance, SAWS cannot provide regular weather reports for all South African towns without incurring a large financial cost since there are a large number of them (there are 3109 mainplaces, a geographical classification that refers to a city, town, or large area [278] in South Africa). Similarly, if some South African universities wanted to provide students with very detailed progress reports throughout the year then they would need to hire specialised staff and incur the costs since they cannot expect a lecturer to compile such reports, especially since some courses have a very large number of students [119].

1.1 Computer systems for text generation

Natural Language Generation (NLG) solutions can overcome the scalability challenge. Specifically, one can build and deploy a data-to-text system for the previously mentioned examples. For instance, the United Kingdom's Meteorological office[2] launched

[1] `https://www.weathersa.co.za/`
[2] `https://www.metoffice.gov.uk/`

an NLG system that can generate forecasts for approximately 5000 locations automatically [260]. Another example of such a system is Isard and Knox's [117] student report card generator that was deployed for three modules of a Master's degree at the University of Edinburgh over three semesters [141]. NLG is a subfield of Natural Language Processing (NLP) and is concerned with researching and modelling various language phenomena and building software systems that can generate text from data, information, or knowledge. The creation, and deployment, of NLG systems is not limited to research-focused institutions as there is a growing number of commercial companies that sell NLG technology and related services [66, 67]. Prominent examples of companies that are operating in this space are Arria[3], Ax Semantics[4], and Yseop[5]. An example of the deployment of an NLG system outside a research and teaching enviroment is the use of Ax Semantics' technology[6] by News.de[7], a German independent news portal, to create daily reports of current news, statistical reports pertaining to the Coronavirus disease, etc.

NLG software systems are responsible for determining document structure, word phrasing, sentence combinations, and the correct surface form of the linguistic units (e.g., symbols, words, or sentences) with respect to the chosen natural language's orthography. Some of these systems tend to generate text following the two-step process depicted in Figure 1.1.

Figure 1.1: High-level depiction of the process of generating natural language text from some input.

The planning component in the above process is in charge of picking the input portions that are judged relevant and deciding the organisation of those pieces. Then, the realisation component converts a system's intermediate representation to a natural language. While the planning process might be natural language agnostic, up to a

[3]https://www.arria.com/
[4]https://en.ax-semantics.com/
[5]https://www.yseop.com/
[6]`https://en.ax-semantics.com/case-study/newsde/`
[7]`https://www.news.de/`

point, the realisation component is always language-specific. Several tools have been built to support languages such as English [94], Tibetan [146], and French [200]. We will discuss them in detail in Chapter 2. However, there are no easy to maintain and reusable tools built for isiZulu and isiXhosa, the most widely spoken Nguni languages. Before we delve into great detail about the limitations of the few tools created for isiZulu and isiXhosa, we will introduce these two languages.

1.2 Nguni languages

The Nguni language group belongs to the Niger-Congo B (NCB)[8] language family and is composed of languages spoken primarily in Southern Africa. The exact number of Nguni languages is dependent on the NCB language classification since some may be genetic vs. referential. The genetic classification systems use historical discoveries from fields like archaeology or genetics to make conclusions regarding language similarity. The referential classification systems rely on the comparison of language features and do not draw a conclusion regarding a language's history [250]. In addition, some classification systems may use different methods to distinguish between a language and dialect. In this thesis, we shall use Maho's [173, 174] system since it is the most recent classification of NCB languages published in English, a language understood by the present author. According to the classification, the Nguni group has the following languages and dialects (we explain it afterwards):

- Xhosa (S41):
  - Mpondo (S41A)
  - Xesibe (S41)
  - Bomwana (S41C)
  - Gaika (S41D)
  - Gcaleka (S41E)
  - Thembu (S41F)
  - Mpondomise (S41G)
  - Ndlambe (S41H)
  - Hlubi (S41I, in the former Ciskei)
- Zulu (S42):
  - KwaZulu-Natal Zulu (S42A)
  - Transvaal Zulu (S42B)
  - Qwabe (S42C)
  - Cele (S42D)
- Swati (S43)
- Ndebele of Zimbabwe (S44)
- Old Mfengu (S401) [*This language is extinct*]
- Bhaca (S402)
- Hlubi (S403)

[8] Elsewhere in the literature, they are called the Bantu language family. We avoid this term due to the derogatory nature of the word "Bantu" in South African history.

- Phuthi (S404)
- Nhlwangwini (S405)
- Lala (S406) [*This language is extinct*]
- South Ndebele (S407)
- Sumayela Ndebele (S408)

In the list of languages given above, we have kept Maho's orthography for the various language names (e.g., we use Zulu instead of isiZulu). We also kept the language codes to enable easy identification of each language's geographical predominance using a coded map of NCB languages (e.g., [173]). This list uses a language code that ends with a capital letter to denote dialects. All other items are languages (e.g., Xhosa, Zulu, Swati, etc.). The list also uses the terms 'Transvaal' and 'Ciskei' — references to the province and territory that existed in Apartheid South Africa[9]. Please note that some authors may classify *Old Mfengu*, *Bhaca*, *Hlubi*, *Phuthi*, *Nhlwangwini*, *Lala*, *South Ndebele*, and *Sumayela Ndebele* as dialects of either *Xhosa*, *Zulu*, *Swati*, or *Ndebele* but Maho classifies them as separate languages since he is of the view that the "closest affinity within the group is uncertain or impossible to decide" [174, pg640].

While Maho's list of languages includes numerous Nguni languages, this thesis focuses only on isiXhosa and isiZulu.We have chosen the two languages since they have the highest number of first language speakers in the Nguni group; as such, NLG support can impact a vast number of people. Specifically, there are around 19 million first-language speakers of the two languages [261].

The languages have agglutinative morphology, a complex verb structure, noun classification, and concordial agreement. We use the example from [167] to illustrate the verb's complexity and various aforementioned features:

ba-sa-si-neth-isis-a
3pers pl-ASP_{P}-OC-rain_{VR}-INT-FV
'It is still raining intensely on us as a result of them'

The verb is formed by combining several morphemes. In addition, the *ba-* is used to indicate the 3rd person plural and the *-sa-* is the perfective aspect. The *-si-* is an object concord used to indicate that the verb's object is the 1st person plural. The object concord is dependent on the noun class of the verb's object. There are 15 and 17 noun classes for isiXhosa and isiZulu, respectively. However, some noun classification systems may have fewer classes. The verb root is *-neth-*, *-isis-* is a verb extension and

[9]For more context on the geographical areas, see `https://www.sahistory.org.za/article/black-homeland-citizenship-act-1970`

denotes the intensive, and finally *-a* is the final vowel. The noun classification system can be understood as being similar to the ones found in languages like Mandarin [161] or Amazonian languages [77]. Nonetheless, it manifests itself in different ways in NCB languages. For instance, in the above examples, the verb has morphemes whose values depend on their governing noun.

1.3 Generating NCB languages

At the time of writing, there are only a few NLG systems [138, 42] that generate languages from the NCB family. While the subset of the NLG community that focuses on languages like English has embraced the use of deep learning [206], this has not been the case for NCB languages owing to a lack of data. Instead, researchers who focus on the languages under consideration have created systems that are limited to specific application domains as there are no surface realisers that can be used with NLG systems whose input is data, information, or knowledge. The creation of neural machine translation systems that take in data or knowledge as input to generate English text that can then be translated into isiZulu or isiXhosa cannot, at present, lead to systems whose quality is at the level of languages like English (e.g., a good English-to-IsiZulu system has a BLUE score of 7.54 [183] vs. a score of 30.2 in a good English-to-German system [162]). This is because such an approach assumes the existence of large parallel English-IsiXhosa and English-IsiZulu corpora. However, such corpora do not exist, at least not to be able to build high quality translation systems. This situation is even worse when we consider the availability of data in such sub-domains such as data-to-text and knowledge-to-text (i.e., it is often easier to collect parallel corpus for machine translation vs. a data-to-text corpus for the languages). Technically, one can still create subject-domain independent systems using some of the realisation methods used by the existing systems (i.e., patterns). For instance, Keet and Khumalo's patterns can verbalise part-whole relations in any ontology, irrespective of its subject domain. Similarly, Byamugisha et al.'s [42] patterns and algorithms can verbalise Web Ontology Language (OWL) constructors for other domains even though they were designed to support the healthcare domain. To support the efficient creation of NLG systems, especially including ones that also take other kinds of input other than ontologies, there is a need for reusable and easy-to-maintain surface realisers for the languages. Such realisers would reduce the need to re-engineer text generation tools that are applicable in most, if not all, subject domains when building an NLG systems.

An example of a reusable tool for surface realisation, even though it only supports English and not languages in the NCB family, is SimpleNLG [94]. The realisation engine was used in the Hotel Scribe system [165], for generating English hotel descriptions from structured data. It has also used in other systems outside the hotel domain (e.g., [91]). In addition, there are also variations of SimpleNLG built to support languages like German [28], Galician [90], and Italian [235]. Since South African languages are under-resourced [99, 201], there is no equivalent grammar-based (or even data-driven) realisation solution. It is also not feasible to develop large corpora or write grammar textbooks from scratch prior to building an NLG system for each domain. A lot of money [216] and time is required to create such resources; hence, other techniques for surface realisation are sensible for the languages.

All existing NLG systems that generate NCB languages use patterns [136]. These are an extension of the notion of 'simple templates' that introduce sub-lexical elements, rules for word agreement, and orthogonal Computational Grammar Rules (CGRs) for phonological conditioning. The latter refers to the rules that are required for combining a word's morphemes and will be discussed in detail in Chapter 2. Our use of the term CGR is meant to clarify that the grammar rules to which we refer are represented in a way that would enable a computer to process them. We also use the term 'simple templates' to refer to templates with fixed words and elements only. We will introduce them in detail in Chapter 3. The following pattern is a fragment of the patterns created by Keet and Khumalo [136] for verbalising universal quantification:

⟨ `QC (all) for` NC_1⟩`onke` ⟨N_1⟩

The pattern has a word whose prefix is a slot and whose suffix is the string *-onke*. The second item is a slot that expects a noun (N_1) from a specific noun class (NC_1). The first word's slot expects a universal quantitative concord (*QC (all)*) that is dependent on the noun class of the word inserted into the second item. For instance, the pattern can be used to produce the texts *bonke abantu* 'all people' and *zonke izinja* 'all dogs' where the underlined segments are determined by the inserted noun. The slot with the universal quantitative concord is not possible in traditional templates as the only permissible slot type is the one used for the noun (i.e. ⟨N_1⟩). This is because it exists within a word and its value is dependent on another word. In the given example, the noun to be inserted into the slot controls the form of the preceding word and this process may appear similar to numerical classifiers in Mandarin. It differs in that NCB nouns can be classified into more categories and they impact the form of many

more Part-of-Speechs (POSs). The categorisation of nouns for Nguni languages will be detailed in Section 2.1.

The use of CGRs within templates is not unique to NCB languages. For instance, RosaeNLG offers support for so-called augmented templates for English where some linguistic rules are applied to templates. Consider the augmented template shown in Listing 1.1 (we explain it afterwards):

Listing 1.1: RosaeNLG template for displaying a list of items.

```
1  mixin someExampleMxn
2    -
3      const items = [
4        ['sword', 6],
5        ['element', 1],
6        ['corpus', 10],
7        ['dog', 1]
8      ];
9    | There are multiple objects we use for demonstration, for example
10   eachz stuff in items with {separator: ',', last_separator: ',
11       plus', end: '!'}
12     -
13       const name = stuff[0];
14       const num = stuff[1];
15     if num > 1
16       | #[+value(num, {'TEXTUAL':true })]
17     else
18       | a
19     | #[+value(name, {number: valueToSorP(num)})]
20 p #[+someExampleMxn]
```

The above augmented template generates the text "There are multiple objects we use for demonstration, for example six swords, an element, ten corpora, plus a dog!". Specifically, it generates an html paragraph using the mixin someExampleMxn (line 20), whose body is defined in lines 1-19. A mixin can be understood as reusable code that resembles a function. The underlined segments are generated by the value mixin[10] (lines 16 and 19) and it produces texts of the form '$NUM\ NOUN_{pl}$' or '$DT/NUM\ NOUN_{sg}$' where NUM and DT denote a verbalised Indo-Arabic numerical and a determine, respectively. The first text form (i.e., $NUM\ NOUN_{pl}$) is used when verbalising multiple items (e.g., 10 corpora) and the second form (i.e., $DT/NUM\ NOUN_{sg}$) is used when verbalising a singular item (e.g., 1 dog). The value mixin uses CGRs to verbalise Indo-Arabic numerals (NUM) and generate the determiner (DT) that morphologically agrees with its corresponding noun. In the English augmented template, the rules are not 'augmented' directly within the template. Instead, the template

[10] https://rosaenlg.org/rosaenlg/3.1.0/mixins_ref/value.html

specification makes use of a value mixin whose operations is defined in RosaeNLG's ValueManager. Nonetheless, a tool suite like RosaeNLG is not suitable as foundation for building reusable and easy to maintain systems for NCB languages.

RosaeNLG and related tools will be discussed in detail in Chapter 2 where we will also point out their limitations.

1.4 Limits of existing surface realisers

To demonstrate the limitations of existing surface realisation techniques that have already been explored for NCB languages on a broader level, consider the following example scenario:

> A South African banking company whose customer base is isiZulu and isiXhosa speaking members of burial societies and stokvels has amassed a large amount of financial data over several years. They have used it to identify the need to reduce customer visits to physical branches for certain matters. In addition, they are interested in generating reports for their customers to use. The reports will be delivered via smart-phone applications to help their customers reduce costly behaviours. After consulting their engineering department, they have decided that they will introduce specialised conversational agents and NLG report generators in Nguni languages.

It is reasonable for the engineers described in the scenario to expect that they could also rely on the patterns used by Keet and Khumalo [136] and Byamugisha, Keet, and DeRenzi [44]. In addition, it is also reasonable to expect that there are modular surface realisation tools usable in their two application areas. Such modular tools would reduce the duplication of effort. Since their application areas are not knowledge-to-text then they would need to adapt or extend patterns. There are two existing approaches for making use of patterns for domains or languages for which patterns [136] were not designed:

- Extension: This approach involves creating additional constituents in order to support new domains, when necessary. This approach was seen in Keet and Khumalo [134]'s extension of the original patterns. They amended the patterns' original constituents with the possessive concord and locative affixes to support the verbalisation of part-whole relations in isiZulu. We use RosaeNLG

to demonstrate how this approach has also been seen for template specifications that are designed for English. The template specification introduced in Pugjs'[11] template engine was extended by introducing the syn_fct function, among other additions.

- Adaptation: This approach involves using patterns for NCB languages, other than the ones for which patterns were initially designed. Byamugisha, Keet, and DeRenzi [44] used the approach to build support for Runyankore, a Ugandan language belonging to the NCB family albeit in a different group than isiZulu — Keet and Khumalo's [136] target language. Their approach involves introducing new values for concords and affixes. It also introduces rules for specifying the order of words in Runyankore. To an extent, we can observe the adaptation process for English in Kuanzhuo, Lin, and Zhao's [146] creation of SimpleNLG-TI based on the original SimpleNLG [94]. Specifically, they made use of the categories already presented in SimpleNLG. However, they also introduce a novel "rule bank [that] includes orthography, morphology and syntax rules [extracted from] Tibetan grammar books" [146].

Both approaches have not focused on architectural considerations to produce reusable and easy-to-maintain surface realisation tools. They duplicate grammatical agreement rules in their generation algorithms instead of creating mechanisms of allowing reuse of the same scant resources. This duplication is a result of patterns' tight coupling of the text generation rules with the template. The adapted/extended patterns, similar to the original patterns, do not have an explicit declaration of their concepts and constraints. Consequently, it is impossible to use computational methods to detect invalid templates (i.e., ones that violate the declared constraints). There are also no existing surface realiser architectures that can be relied on to produce maintainable template-based realisers. We recognise that at first consideration, it may seem like the reliance on templates in a maintainable realiser is contradictory, but we concur with van Deemter, Theune, and Krahmer's argument that even if adapting such realisers to a new domain requires the development of new templates, such architectures can still be maintainable since the "underlying generation mechanisms generally require little or no modification" [273].

The limitations and challenges noted above are a result of the following reasons:

- not having a systematic manner of associating templates with CGRs;

[11] `https://pugjs.org/`

- not having a specification of templates that supports interoperability and the use of existing computational resources for error detection (e.g., a rules engine); and
- not investigating how best to organise surface realiser modules and artefacts, especially with regards to how maintainability and reuse in mind.

The approaches described in [134, 44] neglected these issues because the authors were investigating different research questions. Specifically, Keet and Khumalo [134] were focusing on the sufficiency of simple templates for generating isiZulu text since the language has complex grammar. Byamugisha [44] was focusing on the development of techniques to bootstrap the isiZulu resources for another NCB languages and determine the generalisability of said techniques.

A tool like RosaeNLG is also not viable. The tool was created to make it easy to get started for engineers with no formal background in NLG. This is most likely why it was built on top of a popular web templating system[12]. Its simplicity comes at the cost of expressiveness, it uses *ad hoc* approach for pairing templates and rules, and offers no linguistic support for any African languages.

There is a need to investigate the foundations for modular surface realisation to support cases like our example scenario. By foundations, we are referring to improved approaches for pairing templates and CGRs, specifications of templates that support interoperability, and an architecture for easy-to-maintain surface realisers.

1.5 Problem statement

To the best of our knowledge, there are no systematic and planned methods of associating templates with CGRs. There are also no specifications of interoperable templates, especially ones that have support for morphologically rich languages. In addition, there are no architectures for creating an easy to maintain and reusable template-based surface realisers. As such, there are no Nguni language surface realisation tools that are easy to maintain, can be used for knowledge-to-text and data-to-text systems, and generate understandable and grammatically correct text.

This problem is significant because governmental organisations such as the SAWS can use such tools to generate weather reports. Commercial companies can use them to build a variety of NLG systems to solve their business needs. More importantly,

[12] `https://pugjs.org/api/getting-started.html`

they can do so without the need to recreate the basic building blocks of these systems every time. It is currently impossible to achieve this because existing surface realisation methods used for other languages are not viable for Nguni languages. Simple templates are not fit for purpose because they cannot capture agreement. Existing template extensions (i.e., patterns and so called augmented templates) are *ad hoc* and have several limitations. There are no grammar engines for the languages as the existing CGRs are limited in scope, and data-driven approaches are not viable due to lack of corpora.

1.6 Research questions and tasks

This work aims to establish the foundation for reusable and easy-to-maintain and reusable surface realisers for Nguni languages. We have set out to achieve this by investigating the three main research questions given below. The questions are sensible because they address a gap in the theoretical knowledge required to create engineering solutions that collectively solve the research problem.

RQ1 What are the characteristics of systematic methods of pairing templates and computational grammar rules to form augmented templates? How do they enhance the selection of models of templates to support grammatically complex languages?

(a) How to relate templates with CGRs to support template scaffolding and resource reuse?

(b) How do the *ad hoc* models used by NLG existing systems differ concerning their support for the features mentioned in sub-question 1a?

RQ2 What are the various surface realisation tasks and how can they be organised to produce surface realisers that are reusable and easy to maintain for Nguni languages?

(a) How do surface realisation tasks, as found in existing NLG architecture comparisons, limit the analysis of existing surface realiser architectures at finer granularity concerning their suitability for Nguni languages?

(b) What are the granular tasks, low-level than the tasks identified in part 2a, and how can they organised to achieve easy to maintain and reusable surface realisers for Nguni languages?

(c) What are the algorithm(s), template specification(s), and annotation model(s) that are needed for the relevant tasks mentioned in question 2b in order to produce correct text?

RQ3 Do Nguni NLG systems that use a realiser that organises its modules according to the method established in **RQ**2 generate text that is correct (i.e., fluent, acceptable, and/or grammatically correct) in selected data-to-text and knowledge-to-text scenarios?

In order to answer the research questions, we devised the following tasks:

Task 1 Develop a model-based approach to pairing templates and grammar rules.

Task 2 Develop an architecture to be used when organising surface realisation components for template-based realisers that are easy to maintain.

Task 3 Create a task ontology to support the capturing of simple templates that support morphologically-rich languages and achieve interoperability in template-based systems.

Task 4 Develop proof-of-concept implementations of algorithms and tools required to achieve the tasks of the architecture's modules.

Task 5 Evaluate the sufficiency of the developed approaches and artefacts for generating fluent/acceptable and grammatically correct isiZulu and isiXhosa text.

At the end of the chapter, we will detail how each task was conducted and also provide a guide on which chapter provides further descriptions of the various tasks. We now turn to contextualisation of the tasks to clarify how they fit together in a text generation scenario. We will also specify which task will answer each research question(s).

1.7 Contextualisation of tasks

To demonstrate the relationship between the various tasks, consider the informal visualisation in Figure 1.2. We label various parts of the figure with the numbers corresponding to the tasks in a text generation scenario.

Task 1 pertains to pairing the simple templates and computational grammar rules. It focuses on the existential relationships used to combine the two assets. For instance, understanding the relationship between some template t and CGR g, would

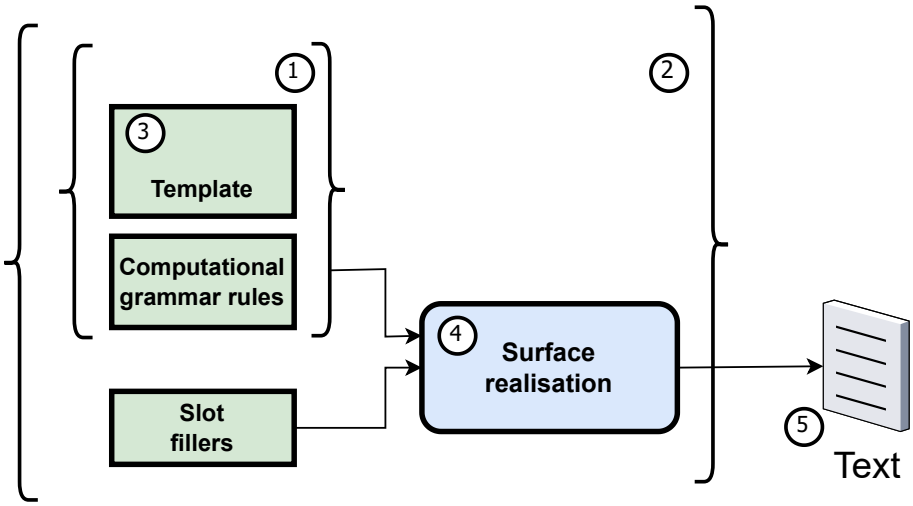


Figure 1.2: Informal rendering of the relevant components involved in the surface realisation task and the numbering of elements that pertain to the identified tasks

make it possible to determine whether g is usable for other templates when t is removed/deleted. We now turn to demonstrate this task's importance using RosaeNLG[13]. When we examine the approaches for encoding rules for synonymy in RosaeNLG, we see that they can be captured using `syn_fct` function or `synz-syn` mixins. The choice between the two possibilities has implications for the reusability of the rules. This task's goal is to offer a foundation for understanding such differences. The task will answer research question 1 and its sub-parts.

Task 2 focuses on the organisation of the various realisation tasks and their corresponding modules. It builds upon task 1 by identifying various realisation tasks, technologies required to achieve them, and figures out how they must be organised to achieve maintainable realisers. The purpose of this task is to make clear the various differences that are either taken for granted or unknown from an architectural perspective. For instance, the publication that introduced SimpleNLG [94] drew a distinction between a realisation engine vs. tactical generation. In addition, Abed and Reiter [2]'s publication describing Arabic language functions alluded to there being differences between RosaeNLG and SimpleNLG. However, these differences have not been explicitly codified in the main. If they are not considered when building NLG systems, the resulting system may be negatively impacted from an architectural perspective. This task will answer research question 2 and its sub-parts.

Task 3 focuses on the method used for specifying the templates, especially for sup-

[13] `https://rosaenlg.org/rosaenlg/3.2.1/`

port morphologically rich languages. Continuing to use RosaeNLG for demonstrative purposes, we see that one can extend the notion of templates (e.g., by introducing mixins for capturing linguistic rules as part of the template) and use a specific language to capture such templates (e.g., Pugjs' custom template language). A tool like RosaeNLG, since it is built to support well-resourced languages, can afford to make the synonymy mixins, and possibly other kinds of linguistic rules, as part of the template in a random or unprincipled manner. However, low-resourced and morphologically rich languages must be deliberate regarding what kinds of components and rules are part of the template vs. the ones that are separate but used by the templates. This task will answer research question 2 part (c) by determining the requirements and creating a task ontology for template specification.

Task 4 pertains to the creation of all the resources required for the various modules found in the architecture. It will answer research question 2 part (c) as its focus is building the identified algorithms and annotation models. Task 5 involves evaluating the quality of the texts and creating a variety of proof-of-concept tools to support the generation of text in isiXhosa and isiZulu. The task will answer research question 3.

Please note that Figure 1.2 is only illustrative and does not specify what components should be in the realiser. It also does not commit to any specific organisation of the augmented templates and slot fillers. For instance, while the computational grammar rules are represented as being outside the realiser, that need not be the case in the final architecture to be investigated and created. We now turn to discuss the approach we have taken in this thesis.

1.8 Research approach

The research approach followed to execute the tasks and answer the research questions is structured into two main steps: (1) development of a model-based pairing of templates and CGRs, a task ontology for templates, an architecture for maintainable template-based surface realisers, and implementation of the architecture, and (2) evaluation of the solution concerning the scope and ability to generate understandable and grammatically correct Nguni language text. In the following paragraphs, we elaborate on each of these steps:

Solution development We follow a three-step process aimed at producing (i) a method of pairing templates and CGRs, (ii) an ontology for template specification, and (iii) a surface realisation architecture that will lead to maintainable tools to solve the research problem. Specifically, the first step focuses on the theoretical foundations of pairing templates and CGRs. This involves the creation of a model of pairing the two artefacts and using it to devise a classification scheme for identifying the various types of templates that come about when simple templates are paired with CGRs. We demonstrate the utility of the scheme by classifying all existing templates that are paired with CGRs, as they are used by as many existing NLG systems as we could find. The second step focuses on the creation of a template ontology called Task ontology for CNL Templates (ToCT), formalised in the OWL, to allow the capturing or serialisation of morphologically-rich templates meant to support Nguni languages. We determined and captured the requirements of said ontology prior to development, following best practice. After the creation of the ontology, we verified that it captures the requirements by determining the answerability of the SPARQL Protocol and RDF Query Language (SPARQL) queries that translated from the Competency Questions (CQs). The third step focuses on the architectural considerations of building knowledge-guided NLG systems that rely on the created model of grammar-infused templates. More precisely, we identified the key tasks located in and around the surface realiser in several existing systems. We used them to create categorisations of the architectures used by existing systems. That allowed us to identify and fill gaps for a maintainable architecture that is appropriate for template-based surface realisers. Moreover, we also developed an implementation of a surface realiser that abides by said architecture.

Evaluation We focus on the engineering of data-to-text and knowledge-to-text systems for isiZulu and isiXhosa in the final step. We evaluate and demonstrate that the created surface realisation solution offers advantages over simpler techniques. In addition, it is capable of generating fluent and grammatically correct text. While we have evaluated the model-based approach, task ontology, and architecture separately, the purpose of the final evaluation is to demonstrate that the developed approach and artefacts, when used together, generate texts of good quality.

1.9 Thesis outline

The rest of the thesis is structured in the following way:

Chapter 2: This chapter mainly focuses on a detailed problem demonstration. We show that the surface realisation techniques for other languages are not sufficient for the languages in question. Specifically, the chapter begins by providing background for Nguni languages. It then draws attention to the features that need to be taken into account when building NLG solutions for these languages. After that, we provide a brief overview of NLG, with a specific emphasis on data-to-text and knowledge-to-text systems built for African languages. Lastly, we discuss existing surface realisation solutions and their limitations in the context of Nguni languages and use the introduced features to demonstrate their gaps.

Chapter 3: In this chapter, we introduce our model-based approach of pairing templates and grammar rules. Specifically, we present a mathematical definition of simple templates, associable grammar rules, and grammar-infused templates. Based on the relations used between simple templates and CGRs, we demonstrate the categories of template types that arise. We then classify existing template-based systems in one of the template categories to get a better picture of their differences regarding CGR reuse and supported natural languages, among other things. Lastly, we demonstrate the use of the classification of existing systems. The proposed model-based approach of pairing templates and CGRs assists when deciding how to build new template types from scratch. In addition, it is also useful when selecting a candidate from the existing methods. We demonstrate this via a use case that focuses on picking an appropriate template type for supporting isiZulu text generation.

Chapter 4: In this chapter, we create an ontology-based specification of templates that also supports morphologically rich languages. We determine the requirements for a model of templates using our domain knowledge regarding English-oriented templates and Nguni languages. We then develop the ontology by relying on a variety of primary knowledge sources to meet those requirements. We then demonstrate that the developed ontology satisfies our requirements by using the standard method of relying on SPARQL queries. Lastly, we demonstrate how to capture isiZulu and Catalan templates using the ontology.

Chapter 5: In this chapter, we develop a novel architecture for maintainable surface realisers. We design the architecture with special attention for natural language generation systems that produce low-resourced languages, notably those with agglutinating morphology. We begin by analysing the architectures used by

existing surface realisers. Our analysis focuses on choosing sentential structures, the method used to encode said structure, and the location of the rules used to generate text from the structure. The importance of these features impacts complexity (in the case of location of the structure selection) and reusability (in the case of linearisation rules) in the resulting surface realiser. This process identifies gaps in existing architecture with respect to maintainability, specifically concerning under-resourced languages. We then present a new architecture to address the identified gaps, compare it with existing architectures, and discuss the utility of using an ontology to capture template knowledge in the architecture.

Chapter 6: In this chapter, we investigate the quality of isiXhosa and isiZulu texts through surveys. First, we describe the various proof-of-concept tools and artefacts needed to create the two tools used to generate the texts. More specifically, we present two ontologies for annotating concord types, an implementation of a grammar engine, and an implementation of the architecture introduced in the previous chapter. We then present an isiXhosa weather forecast generator and an isiZulu question generator for evaluating the developed approaches and artefacts. Both generators use templates captured via the ontology presented in Chapter 4 and their surface realiser abide by Chapter 5's architecture. We demonstrate that most of the texts generated by the systems are judged positively by survey participants. In addition, the templates include elements that are outside the scope of English-oriented templates.

Chapter 7: We conclude by demonstrating how the various artefacts, that have been introduced and evaluated in the previous chapters, collectively solve the research problem. Moreover, we also present some areas that are interesting as future work.

Appendix A : We present the competency questions and their associated SPARQL queries that are used in Chapter 4.

Appendix B We present the materials that are associated with the two surveys presented in Chapter 6. We have included a human-friendly version of the input and output from the isiZulu knowledge-to-text system, the survey instructions presented to participants in all the surveys, and the ethical clearance letter for the survey.

Chapter 2

Background

This chapter aims to provide background knowledge regarding the languages we are focusing on, i.e., isiZulu and isiXhosa, and the field of natural language generation. It also discusses the key literature whose focus is surface realisation, with emphasis on methods used for African languages and their limitations with regard to the research problem.

We will begin by discussing isiZulu and isiXhosa's noun classes, their concord-based agreement systems, and how to apply the necessary grammatical rules when forming words. We will also discuss the impact of all such features on natural language generation. In Section 2.2, we will discuss natural language generation in general terms and also discuss the internals of the Natural Language Generation (NLG) systems that generate African languages. In Section 2.3 we focus on the architectural design considerations for NLG systems, even when said systems are generating non-African languages. Lastly, in Section 2.4 we delve into how existing surface realisers are created. This entails a discussion of existing architectures and an examination of how existing realisation techniques are classified. We refine the categorisation and classify existing techniques anew. The process brings mixed-methods concealed by high-level categorisations to light.

2.1 IsiXhosa and isiZulu

IsiXhosa and isiZulu are members of the Nguni language group. There are four languages in the group (i.e., isiXhosa, isiZulu, isiNdebele, and siSwati). The languages are mostly spoken on the eastern coast of South Africa, as can be seen in Figure 2.1.

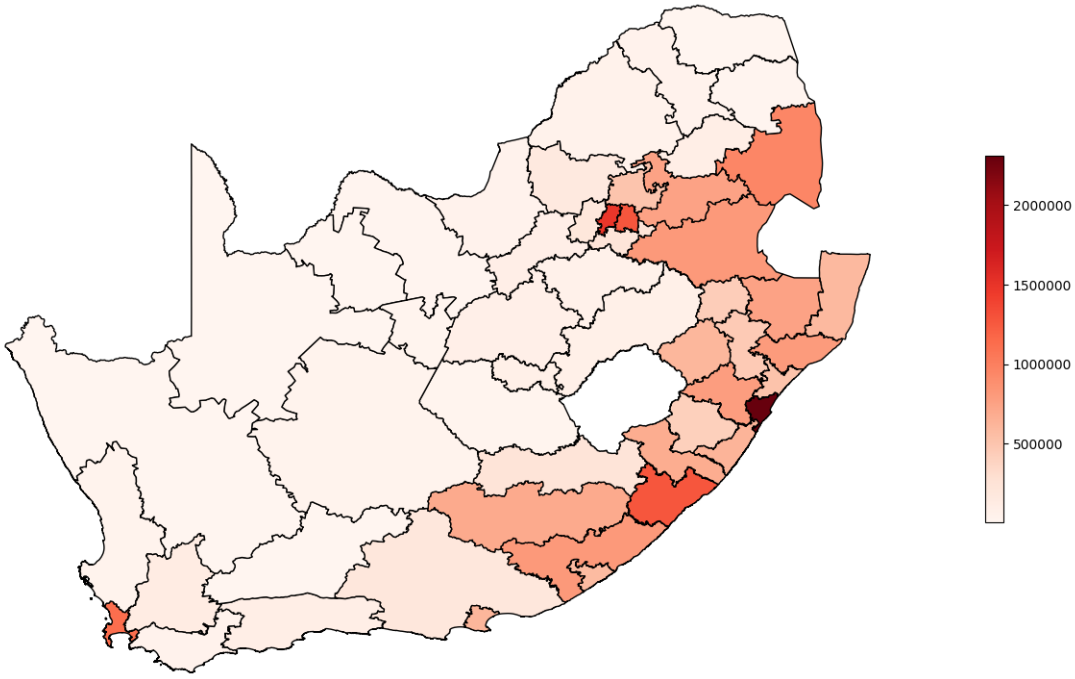


Figure 2.1: Number of IsiZulu/IsiXhosa speakers for each district municipality in South Africa, per 2011 municipal borders specified by the Municipal Demarcation Board (`https://dataportal-mdb-sa.opendata.arcgis.com/`). The number of speakers is computed using the 2011 Census Community Profiles [262].

IsiXhosa and isiZulu are the largest languages out of all four, by the number of L1 speakers. They are collectively spoken by over 19 million L1 speakers [261].

The two languages, like other Niger-Congo B languages, have agglutinative morphology and an agreement system. This means that words are formed by concatenating morphemes together. In addition, some words have morphemes whose values rely on other words in the same sentence. For instance, the isiXhosa equivalent of 'the people have left' is *abantu ba**mkile*** where the underlined part is an affix, called the subject concord, that is associated with noun class 2. The affix is used in verbs to denote that its subject is a noun from class 2 (i.e., *abantu*). In the sentence *izinja zi**mkile*** 'the dogs have left', the verb root from the previous example has a different subject concord (*zi-*) since the noun that acts as the subject, *izinja*, is from noun class 7 instead of 2. There are numerous concord types and several noun classes for each Niger-Congo B language. In this section, we will discuss these characteristics in detail.

Table 2.1: List of noun classes and their prefixes in Proto-Bantu (Source: [26, p282-284])

Language	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Proto-Bantu	mŭ	ba	mū	mi	di/li	ma	ki	pi	n	thin	lu	tu	ka	bu	ku	pa

2.1.1 Noun classification systems

Nguni languages, like other Niger-Congo B languages, classify nouns into multiple noun classes. The system is often compared to gender in some Indo-European languages. However, unlike grammatical gender, the noun classification is not correlated to biological sex and the number of classes is more than two. The exact number of classes depends on the language and the classification system used since there are multiple versions. Before we provide the noun classes of the two languages, we first present the noun classification for Proto-Bantu, a theoretical ancestor of all Niger-Congo B (NCB) languages.

A widely recognised classification system was created by Bleek [26]. Its prefixes are given in Table 2.1. Bleek's original classification has been amended 9 times since its creation and the changes were as follows:

1869: Bleek reordered classes thirteen and fourteen [128].

1896: Jacottet reordered classes twelve and thirteen [128].

1906: Meinhof added five additional classes: 17, 18, 19, 20, and 21 [128].

1927: Doke added two subclasses to keep the numbering reverse compatible: 1a and 2a [282].

1967: Cole added two subclasses to keep the numbering reverse compatible: 2b and 8x [282].

1967: Meussen added one new class: class 23 [192].

1971: Guthrie removed class 23 (as listed by [172]).

1974: Welmers added one subclass in order to keep the numbering reverse compatible (i.e., class 6a) and reintroduced class 23 [282].

Changes to a noun classification are not limited to Proto-Bantu. Recently, Byamugisha, Keet, and DeRenzi [46] refined the classification of seven Niger-Congo B

Table 2.2: List of noun classes and their prefixes in isiZulu and isiXhosa (Sources: [282, 294] and [214])

Class	1	1a	2	2a	2b	3	4	5	6	
Zu.	um(u)	u	ab(a/e)	o	oo	um(u)	im(i)	i(li)	am(a/e)	
Xh.	um	u	aba	oo	-	um	imi	i(li)	ama	
Class	6a	7	8	8x	9	10	11	14	15	16
Zu.	ama	is(i)	iz(i)	izi	iN	iziN	u(lu)	u(b/bu)?	uk(u/w)?	pha
Xh.	-	isi	izi	-	i(N)	i(z)i(N)	u(lu)	ubu	uku	pha

languages by introducing 25 new classes while keeping the numbering reverse compatible, based on their prefixes. Their change is not a reconstruction of noun classes. It is a reconfiguration for their seven languages to ensure that noun pluralization is deterministic.

Overall, this shows that there is no unique noun classification. As such, when building NLG systems for the languages in question, one should be able to pick any preferred noun classification. It is not enough to rely on the agreed-upon classification at some specific period since consensus may change. NLG systems need to account for a noun classification's minor reordering (as was done in 1869), extension (as was done in 1906), or reverting to an older version (as was done in 1971). More generally, systems need to be able to interpret how one classification is related to another. This ensures that systems that rely on one classification system do not fall apart when faced with an alternative one.

The noun classes and prefix values, as found in the currently prominent classification, for isiZulu and isiXhosa are given in Table 2.2. We use round braces to denote parts of the prefix that can be eliminated by vowel processes. The capital N (e.g., as found in class 2b in isiZulu) denotes the nasal that may either be *-m-* or *-n-*, and the question mark after braces indicates that the values contained therein are optional (e.g., as used in class 5 in isiZulu). There is no consensus regarding the semantics of noun classes [128] and most authors rely on the rough guide presented in Table 2.3. Early linguists argued that the noun classes are not a semantically motivated categorization [128]. However, several authors have attempted to disprove that for Proto-Bantu [76] or individual Niger-Congo B languages such as Kikuyu [39], Swahili [63], Shona [217], Sesotho, Setswana, isiZulu, and Siswati [75]. Despite these proposals, however, there is no consensus on the matter. Hence, we also rely on Table 2.3 for guidance on interpreting the noun classes.

Table 2.3: Illustrative semantic content of each noun class [128]

Noun Class	**Semantic content**
1, 2	Human beings
1a, 2b	Proper Names, Kinshop terms, Personifications
3, 4	Natural phenomena, Body parts, Plants, Animals
5	Natural phenomena, Animals, Body parts, Collective nouns, Augmentatives, Derogatives
6	*Regular plural of classes 5 and 14*; Mass terms and liquids, Time references, Mannerisms, Modes of action
7, 8	Body parts, Tools, instruments and utensils, Animals and insects, Languages, Diseases, Outstanding people, Amelioratives, Derogatives, Augmentatives, Curtatives (shortness and stoutness), Mannerisms
9, 10	Animals, People, Body parts, Tools, instruments and household effects
11	Long, thin entities, Languages, Body parts, Natural phenomena, Implements, utensils and other artifacts
12, 13	Augmentatives, Derogatives, Diminutives, Amelioratives
14	Abstracts, Collectives
15	Infinitives, a few body parts e.g. arm, leg
16, 17, 18, 23	Location terms
19	Diminutives
20,22	Derogatives, Augmentatives, Diminutives, Amelioratives, Mannerisms
21	Augmentatives, Derogatives

2.1.2 Concordial agreement

IsiZulu and isiXhosa have several morphemes (i.e., sub-lexical items) used to mark agreement between certain parts. They are called concords and there are two categories of these morphemes, namely those derived from nominal and pronomial affixes. The concords derived from each affix category [34] and the Part-of-Speech (POS) categories they affect [192] are listed in Table 2.4.

While Meeussen [192] lists five categories, his numerical and verbal (initial and preradical) affixes are only subcategories of the pronomial. In the table, we continue using the guillemets notation taken from [192]: «**1-5**» refers to the first five cardinal numbers, we assume that «**how many**» refers to quantifiers, «**other**» refers to the demonstrative, and «**which**» refers to the interrogative determiner. There are long and short forms of the *relative* and *adjectival* concords. The short forms exist in isiXhosa but not in isiZulu. Likewise, the *phi/mbi* concords only exist in isiXhosa [34]. The values for each concord type can be found in Bourquin [34]. The concords and other morphemes of a word can be listed following the morphotactical rules of the language but appending them together is not guaranteed to lead to well-formed

Table 2.4: List of Niger-Congo B language agreement affixes and the POS categories they are found in (Source: [192])

Affix category	Concords	Affected POS
1. Nominal	adjectival enumerative	1a. nouns
		1b. locatives
		1c. adjectives
2. Pronominal	subjectival objectival relative possessive phi/mbi	2a. substitutives
		2b. connectives
		2c. possessives
		2d. demonstratives
		2e. determinatives
		2f. «**other**»
		2g. «**which**»
		2h. relative verb forms
		2i. absolutive verb forms
		2j. verb forms
		2k. «**1-5**»
		2l. «**how many**»

words. The reason is that there are morphophonological alternation rules that need to be applied. Sometimes, they are referred to as phonological conditioning/mutation rules. The next section will discuss them.

2.1.3 Phonological conditioning

Phonological conditioning rules are responsible for selecting the appropriate allomorph given some morpheme and context. For instance, when pluralising English words one has to select the appropriate allomorph for the sibilant */s/* as a suffix. For instance, when pluralising the word *test* then one gets *tests* and when pluralising the word *English* then one gets *Englishes*. In the two words, the eventual value of the allomorph */s/* is either *-s* or *-es*.

In the case of isiZulu and isiXhosa, these rules are responsible for eliminating consecutive vowels in the languages under consideration since such vowels "occurring in a single phonological word [are] unacceptable in Nguni [languages]" [252, pg38]. For isiXhosa and isiZulu, we shall consider only the three main categories of these rules; coalescence, gliding, and deletion. Table 2.5 summarises the vowel combinations.

Coalescence occurs when "two identical vowels come together to form a single monomoraic one with the same features and those involving two different vowels whose product

Table 2.5: Combinations of vowels in isiXhosa and isiZulu. Abbreviation(s): IsiZulu = Zu and isiXhosa = Xh. (Sources: [252, 275, 222, 228])

IsiXhosa					
Left \ Right	**a**	**e**	**i**	**o**	**u**
a	a	e	e	o	o
e	ya	e	yi	—	yu
i	i/(ya)	e/(ye)	i	o/(yo)	—
o	wa	we	—	—	—
u	a/(wa)	e/(we)	i/(wi)	o(wo)	—

IsiZulu					
Left \ Right	**a**	**e**	**i**	**o**	**u**
a	a/(wa)	e	e/(yi)	o	o/(wu)
e	ya	e	e	—	e
i	a/(ya)	e/(ye)	i	o/(yo)	u
o	wa	we	we	—	—
u	∅/(wa)	∅/(we)	∅/(wi)	∅/(wo)	∅/(wu)

is a single monomoraic one with non-conflicting features from the two vowels that combine" [252, pg38]. Gliding occurs in "combinations of two vowels where the first in the sequence is" [252, pg40] a high vowel (i.e., *i* or *u*). Lastly, deletion refers to "the dropping of a vowel" [252, pg45]. In Table 2.5, we do not include the transformations that Sibanda [252] includes with question marks as they are speculative. Moreover, use the empty set symbol (∅) to denote the empty string and use round braces for multi-character strings when there is an alternate value for a vowel combination.

We now use the isiXhosa table to demonstrate how the values resulting from the various processes are obtained, depending on their context. Coalescence is partially represented in the diagonal of the table. It is also responsible for eliminating the cases where the left vowel is *a* and the right vowel is either *i* or *u* [252, pg29]. On the other hand, when the left vowel is high or is a mid-vowel then then gliding is responsible for turning it into a semi-vowel (e.g. *i+enza* becomes *yenza* [252, pg40]). However, the following constraints apply:

- In cases where consecutive vowels are preceded by a consonant then glide formation and glide deletion occurs unless the left vowel is *u* (e.g. *si* + *enza* becomes *senza* and not *syenza*).
- In cases where consecutive vowels are preceded by a labial consonant then glid-

ing results in a bilabial, but those are not permitted in Nguni languages. These consonants are the kind that are articulated with either one or both lips. This is resolved using the following two rules: (1) if the bilabial is in the initial section of the word then glide deletion must occur (e.g. *bu* + *enza* becomes *benza*, not *bwenza*) and (2) if the bilabial is non-initial then palatization (See Herbert [107] for more details) must occur on top of the glide deletion [252, pg42] (e.g. *impuphu* + *ana* becomes *imputshana*, not *impuphwana*).

The last process, deletion, is represented using only two cells in the table since it only occurs when the vowel *a* is followed by a semi-vowel. In particular, *a+e* becomes *e* and *a+o* becomes *o*. As such, it can also be captured via simple table-based retrieval.

The discussed features highlight that any NLG system that supports the generation of the Nguni languages needs a flexible mechanism to capture noun classes, morphological dependencies that are due to the noun classification, and phonological conditioning rules to produce well-formed words, in addition to regular care for morphotactics and sentential grammar rules. One must be able to swap the noun classification, should the need arise, since there are numerous versions of noun classifications. Moreover, the combination of vowels needs to be sensitive to the phonological features of their surrounding elements.

2.2 Natural language generation

NLG is a research field that is concerned with building systems that can automatically generate text from a non-linguistic representation of information [68]. The generated text conveys or communicates some specific information to humans. The non-linguistic input may either be data, information, or knowledge. It can also be transformed to other intermediary structures before mapping it onto natural language text. Even when systems take in input that can be classified into a single category (e.g., data), they may still exhibit a number of differences. For instance, while all the systems described by Liu and Lapata [161] and Kasner, Mille, and Dusek [126] are trained in an end-to-end fashion, one system expects text as input while another expects a table. It must also be noted that the aforementioned systems are not modular in the traditional sense and can be used for summarisation, style transfer, fixing source code bugs, etc. Since our focus are (data/knowledge)-to-text systems only, the rest of the section will focus on systems that take in data or knowledge, and exclude all other input types.

NLG systems tend to differ significantly with respect to their architectures and we will discuss that aspect in detail in Section 2.3. In this section, for convenience, we shall introduce NLG systems with an emphasis on modular systems. In the traditional NLG tasks, as illustrated in Figure 2.2, one starts by reading in the input and deciding which information is to be communicated. Then, one decides which concepts should be communicated in the same sentence, choosing words to use for specific concepts, and deciding when to use referring expressions (e.g. "He loves them vs. John loves Jason and Mary"). Lastly, it focuses on converting the abstract representations of the information into natural language words [68, pg49].

We will discuss differences between systems that take in data vs. knowledge as input in the following subsections. This differentiation of NLG systems is of interest because it will be used to demonstrate the wide applicability of the surface realisation techniques to be introduced. We will also discuss the existing work for generating African languages for each of the two types of NLG systems.

2.2.1 Data-to-text generation

The input taken by data-to-text systems is raw data (i.e., sensor data, sports statistics, numerical prediction model data, etc.). These systems offer several benefits when compared to manual report creation. For instance, one can easily produce reports tailored to their target audience without incurring significant additional costs (e.g., [274]) or automatically produce reports for behavioural change while preserving individual privacy (e.g., [37]).

Data-to-text systems tend to follow an architecture similar to the popular three-step pipeline [68] as illustrated in Figure 2.2, but their architectures have an additional step: *signal analysis*. They resemble the architecture used in the BabyTalk project [242] as shown in Figure 2.3. A similar architecture whose first focus is data extraction and concept identification can be found in the older NLG system called Forecast

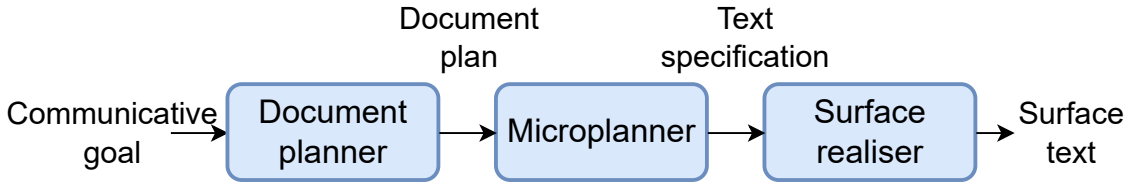


Figure 2.2: Three step pipeline NLG system architecture (Adapted from Dale and Reiter [68])

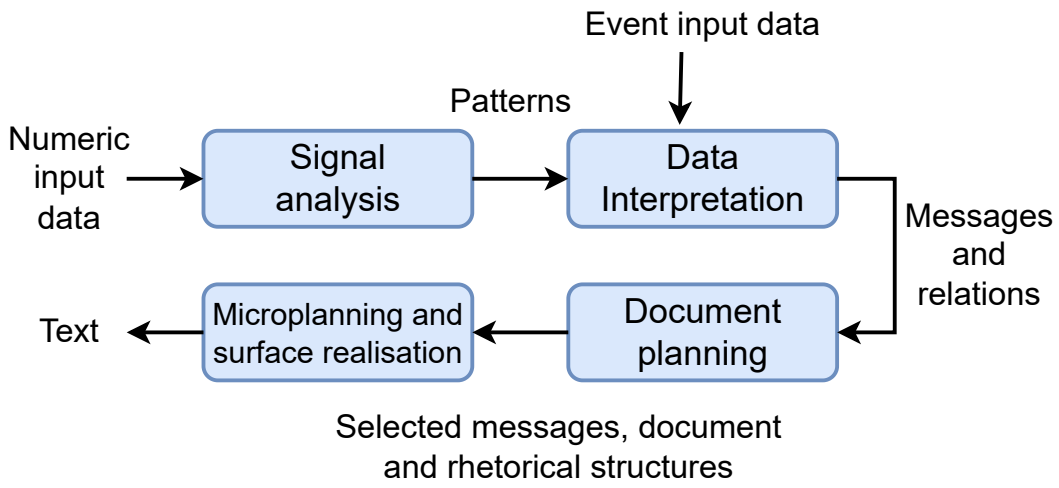


Figure 2.3: Data-to-text system architecture used in BabyTalk (Adapted from Reiter [242])

Generator (FoG) [139]. In that system, weather maps are fed into a conceptual formation module where forecaster edits are possible via a graphical bulletin editor. All of these steps happen before any linguistic processing.

The dominant approach followed by extant data-to-text systems in the research community is to make use of neural networks, even though that might not be the case in commercial enterprises. These systems often decompose the task such that the goal is to map the input data to some vector and then use that vector to generate the output. Most of the early systems of this nature did not introduce a lot of variation to the encoder-decoder architecture popularised by Machine Translation (MT) research [266]. Recently, there has been a growth of NLG systems that abide by the encoder-decoder architecture and introduce an explicit planning component (e.g., [231, 56]). Castro Ferreira et al.'s [51] work even went further by exploring the use of the traditional pipeline architecture where neural models for a number of modules.

Irrespective of the approach taken, there are existing tools and infrastructure to support most processes in an architecture's steps/modules. For instance, numerous algorithms are usable within the signal analysis module (e.g., apriori algorithm for association rule mining). Such algorithms are invariant to the domain area. High-level programming languages support data structures to capture messages and relations required by the data interpretation module. Similarly, there are a number of neural network library that can be used train models for various programming languages. Likewise, all high-level programming languages can easily support the creation of

lexicalisation, aggregation, and referring expression generation rules. Concerning languages like English, human-crafted templates, human-crafted grammar-based systems, statistical approaches, and neural approaches can support surface realisation [93]. While there are numerous tools and methods to support languages such as English, there are only constrained tools that can take text specifications and transform them into well-formed isiXhosa and isiZulu text. We will discuss them in the next section.

2.2.1.1 Existing work on African languages

To the best of our knowledge, there are no data-to-text generations systems built for African languages. There has been limited work in building computational grammar rules for isiZulu and isiXhosa to support the generation of weather forecast verbs [167].

Even though there is no complete data-to-text NLG system for Nguni languages at the time of writing, continuing with a grammar-only approach would not be desirable. It cannot be sufficient for several domains without the labour-intensive task of building comprehensive Computational Grammar Rules (CGRs). CGRs are linguistic rules that are captured in some form that can be processed by a computer. The approach is not sufficient because only a few Niger-Congo B languages have CGRs and those CGRs have limited coverage (see [15, 45, 135, 167, 207, 229]).

2.2.2 Knowledge-to-text generation

Knowledge-to-text systems take ontologies or conceptual models as input [206]. They are used as a means of presenting modelling languages to people who are not knowledge engineers or logicians in an accessible manner, or as modules in Question answering (QA) systems, computer-assisted learning systems, etc. When the input is an ontology, rendering it in natural language is called ontology verbalisation.

The popular three-step pipeline architecture, illustrated in Figure 2.2, is used by most verbalisation systems. In such an architecture, there are modules concerned with deciding which information to communicate, which information should be separated, and which words to use for events. There is also a module responsible for mapping decided-upon abstract structures to natural language sentences. An example of such a system is NaturalOWL [8] and its architecture is shown in Figure 2.4. The system transforms Web Ontology Language (OWL) statements into English and Greek text.

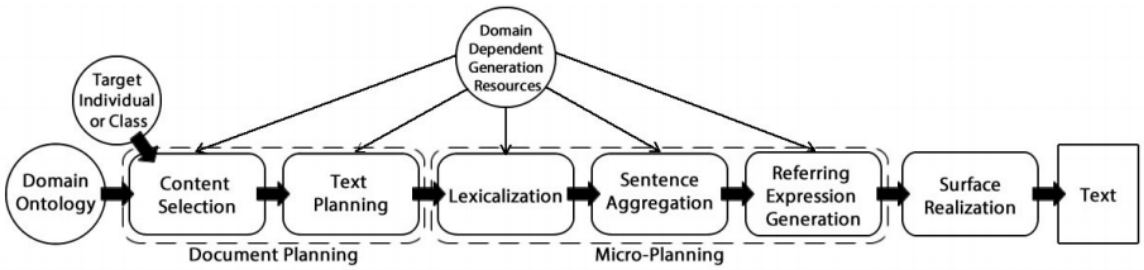


Figure 2.4: Knowledge-to-text system architecture used in NaturalOWL (Source: Androutsopoulos, Lampouras, and Galanis [8])

The above system is not the only kind of knowledge-to-text system. The WebNLG(+) challenges held in 2017 and 2020 led to a number of systems that can verbalise Resource Description Framework (RDF) triples, for instance. In the first challenges description, the following input/output examples where provided:

1. **Input:** (John_E_Blaha birthDate 1942_08_26) (John_E_Blaha birthPlace San_Antonio) (John_E_Blaha occupation Fighter_pilot)
2. **Output:** John E Blaha, born in San Antonio on 1942-08-26, worked as a fighter pilot

In the input given above, there are three triples and the subject of all triples is a person represented using the element called John_E_Blaha. The triples specify that person's birth date, birth place, and occupation in that specific order. The systems entered to the challenges had to find ways of lexicalising the various RDF elements, determining how to combine the segments, and producing the file sentence (among other tasks). It must be noted that the WebNLG+2020 challenge offered additional tasks: generating Russian instead of English and parsing RDF triples from English or Russian text. Overall, these challenges led to a number of international teams developing NLG systems. Some of these systems used the previously described pipeline architecture. However, a number of them relied on a statistical or neural machine translation model. Such systems do not split the generation task into the same subtasks as the pipe architecture. We will describe how they function, with respect to surface realisation, in detail in Section 2.4.

Similar to data-to-text systems, there are tools and infrastructure to support the processes in the various modules in the case of pipeline systems. Similarly, there are a number of software frameworks and libraries that can be used to create Statistical Machine Translation (SMT) and Neural Machine Translation (NMT) models. For instance, there are existing computer-oriented languages to support the representation

of concepts and the relations that exist between them (e.g., OWL). Any high-level programming language can support text planning. There are numerous approaches for surface realisation; namely, templates, statistical and neural techniques, and grammar rules. In particular, there are reusable surface libraries such as SimpleNLG [94] and its adaptations for English, French, Italian, Spanish, Brazilian Portuguese, Dutch, Galician, German, and Tibetan [277, 186, 235, 74, 120, 90, 28, 146]. Unlike the situation for data-to-text systems, there are existing knowledge-to-text systems for African languages. We will discuss them in the following section.

2.2.2.1 Existing work on African languages

Only patterns [43, 136] and templates [249] have been pursued for knowledge-to-text for African languages. In particular, they have been used to build Controlled Natural Languages (CNLs) to support ontology verbalisation for Runyankore [43], isiZulu [136], and Afrikaans [249]. Templates have been used by languages with simple morphology (e.g., Afrikaans [249]). Low-resourced languages that also exhibit grammatical complexity use patterns (i.e., Runyankore [43] and isiZulu [136]). In particular, they combine templates and CGRs such that the CGRs compensate for the inadequacies of the templates, if any, and vice versa. This makes them have wider domain applicability since the constituents of the underlying template should, theoretically, be able to support all domains. Practically, however, the patterns [43, 136] that support Niger-Congo B languages have wholly relied on grammar rules whose scope and pairing with templates is done for specific purposes.

We now turn to explain the purpose of the existing patterns and how they pair templates and grammar rules: the patterns have been created for generating prescriptions from OWL [44, 43] and verbalising OWL ontologies that have specific features, no matter their domain [136]. As a side effect, the patterns lack constituents that are required to support all agreement relations that exist in the Niger-Congo B languages. For instance, the original constituents in the isiZulu patterns (as first presented by Keet and Khumalo [136]) required the amendment of the possessive concord and locative affixes to support domains that require the verbalisation of part-whole relations [134]. In addition, the architecture used by the pattern-based systems is such that the linearisation algorithm is tight-coupled with the template specification. The linearisation algorithm is responsible for inserting values into slots, resolving the shared affixes, and applying phonological conditioning rules, etc. to obtain surface text. As

a consequence, one must re-instantiate that algorithm for each domain for which one builds templates.

In summary, while patterns [43, 136] are the only practical method used to generate text in two different Niger-Congo B languages, they are not geared for general-purpose use. There are also no domain-independent tools for linearising patterns. In addition, unlike well-resourced languages, the use of SMT and NMT has not been seen for the language due to a lack of training data.

2.3 Existing analysis of NLG system architectures

In the previous section, we introduced NLG systems using pipeline architectures. However, not all systems use such architectures. More generally, there are numerous system architectures and they differ based on modularity and the direction of data flow. The architectures are summarised in Table 2.6.

Table 2.6: Classification of NLG architectures (Based on [255, 93])

Integrated	**Modular**	
End-to-end	**Interleaved**	**Unidirectional**
	Interactive	What and How
Planning-based	Blackboard	Three-step pipeline
Classifier cascades	Revision-based	Four-step pipeline
–	–	Generate and select

Systems that do not have any modules dedicated to specific tasks are called integrated systems [255]. There are three kinds of these architectures: end-to-end, planning-based, and classifier cascade architectures. End-to-end architectures are very popular with extant deep learning systems (e.g., [84]). In such architectures, there is a neural network, called the encoder, whose function is to generate a continuous representation from the vectorised input. The encoder's output is then fed into another neural network, called the decoder, that uses the representation to generate other vectorised sequence [206]. While systems that follow this architecture may differ in a number of ways (e.g., some may use a character-to-character model while others may use word-based models), they all have two main 'modules' (i.e., the encoder and decoder). Nonetheless, these two 'modules' are not dedicated to the tasks introduced in Section 2.2 and its not trivial to identify which module NLG enables tasks like referring expression generation, surface realisation, etc. Planning-based architectures are popular with dated systems that viewed generation as a problem of searching for

a sequence of actions, in some action-space, that transforms the initial state (i.e., the input) to some goal state (i.e, the output). Classifier cascades are used in the uncommon NLG systems that treat generation as a classification task (e.g, [182]). The planning-based methods and classifier cascades are not modular in the same sense of the NLG modularity introduced by Dale and Reiter [68]. In other words, their actions or classifiers can "cut across the boundaries of many of the tasks that are normally encapsulated in the classic pipeline architecture, combining both tactical and strategic elements by viewing the problems of ["what to say"] and ["how to say it"] as part and parcel of the same set of operations" [93, p. 86]. The two architectures, unlike end-to-end systems, are not black-boxes as it is possible to interpret the actions or classifiers. Nonetheless, since these non-modular architectures do not have a separate module that transforms their internal specifications into natural language text, it is impossible to isolate and analyse such a module's architecture. If they were modular then it would be possible to identify and resolve problems pertaining to referring expression generation, while keeping the other identifiable modules frozen, and that module can be reused elsewhere as needed. A modular approach is sometimes better than its integrated counterparts since it reduces complexity by making the various dependencies between an NLG systems' tasks visible (cf. a black-box approach are difficult to control [156]).

Modular architectures do have identifiable components that are responsible for surface realisation. These architectures can have two (e.g., [149]), three (e.g., [68]), or four (e.g., [242]) modules. The modules do not only differ in number, they sometimes differ in responsibilities. For instance, Langkilde-Geary and Knight's [149] HALogen system has two modules where the first is responsible for translating the input to a "forest of possible expressions" and selects the most likely output using an n-gram model. Dale and Reiter's [68] 3-step pipeline architecture, on the other hand, does not have a statistical filtering module. Its modules are responsible for document-level structure, word choice and sentence-level, and the application of linguistic rules. Reiter's 4-step data-to-text architecture extends the one proposed in [68] and introduces a signal analysis and data interpretation module for processing the input.

NLG systems that use these architectures come in two categories, interleaved and unidirectional architectures, and the only difference between the two is the direction of data flow. Interleaved systems allow for a back and forth between modules for communicating errors detected by a downstream module back to the upstream module

that caused them. These types of systems are rare due to the engineering difficulty associated with building them.

A popular, but now dated, approach of thinking about modular architectures has been to aggregate the various modules into two classes: those that are responsible for *what* and those responsible for *how* (i.e., "What to say" and the other for "How to say it"). The two module classes have also been referred to using other names and they can be found in [255]. The pipeline architectures with three or four modules [68, 242] are the mature versions of Reiter's consensus architecture [241]. The generate-and-select architecture [149] makes use of mapping rules for transforming the input into multiple possible sentences[1] and ranking via a statistical module.

There has not been an examination and comparison of the realiser component across the different architectures. The only architectural discussion surrounding the realiser is the distinction made between a tactical realiser vs. a grammar engine [94] and demonstration that there is no agreement on what low-level tasks should be in the surface realiser [196] for systems that use the three-step pipeline architecture. Both analyses are limited because while Gatt and Reiter argue for a realiser that is responsible for "spell[ing] out the syntactic constraints of the language in which an utterance is to be generated" [267, p58], they do not make explicit what tasks are required to do so. Moreover, the architecture comparison by [196] only considers three-step architectures and it is high-level with respect to the tasks it considers. For instance, there is no breakdown of the ordering task (i.e., "the choice of linear ordering of the elements of the text" [196, p4]) to identify differences in how realisers achieve it.

To the best of our knowledge, there is limited work that analyses the differences between existing realisation components. Most research that has scrutinised architectural matters only focused on a top-level analysis. In the following section we will discuss the various techniques used for surface realisation in depth. In particular, we will make clear what tasks are achieved by tools such as SimpleNLG [94] and how they compare to related tools.

2.4 Text realisation

Text realisation, or linguistic realisation, refers to the "task of converting abstract representations of sentences into the real text" [68, pg49]. The differences in surface

[1]These rules do not appear to be separated into modules based on the description in the paper but could be since they have different responsibilities

realisation approaches have been examined using the following design choices, some of which are architectural:

- C1: Categorisation into canned text, templates, phrase-based, and feature-based systems [113, pg140-142]
- C2: In-depth vs. surface generation [41]
- C3: Realisation engine vs. tactical generator [94]
- C4: Categorisation into templates, grammars, statistical methods, and neural-based generators [93, 206]

The C1 analysis is a categorisation of realisation approaches into four types. In it, the canned text refers to a fixed string that may be associated with a specific intent. For instance, in a hypothetical thesis submission portal, one may generate the fixed text "You've already submitted your thesis" when a Ph.D. student uploads a duplicate thesis again and that action is not allowed. Templates are fixed texts that have gaps to allow for some variability and reuse. For instance, one could use the template "You've already submitted your [*documentType*]" when notifying the user about different types of documents (e.g., technical report, research paper, and thesis) without creating canned text for each. The phrase-based category refers to methods that rely on phrase structure grammars (e.g., context-free grammars). The feature-based category refers to grammar formalisms that use attribute-values structures (e.g., Systemic functional grammar).

The C2 analysis separates template/canned text-based systems designed for a specific application (surface generators) from knowledge-based systems that have reusable components. The knowledge-based systems are theoretically motivated (in-depth generators). The analysis has been critiqued for its caricature of template-based methods as being simplistic and not linguistically motivated [273].

The C3 analysis relies on understanding the distinction between strategic vs. tactical generation; the former refers to planning while the latter refers to realisation (see [113, 188]). In an NLG setting, planning is about determining "what to say" while realisation is about "how to say it". With that in mind, C3 further divides realisation into two tasks: making tactical decisions and applying routine linguistic operations. Tactical decision making refers to making "appropriate linguistic choices given the semantic input" [94]. Routine linguistic operations pertain to constructing a "syntactic representation, applying the right morphological operations, and linearising the sentence as a string" [94]. For instance, Figure 2.5 shows that the Nigel [177, 178]

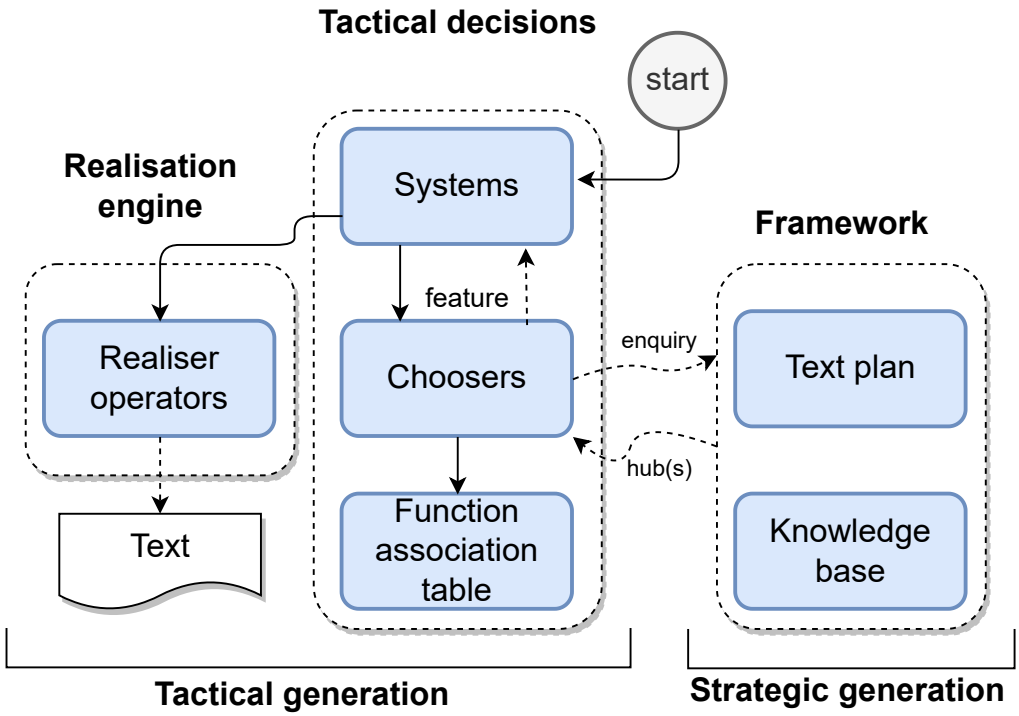


Figure 2.5: Demonstration of the NIGEL grammar within an architecture of the PENMAN project. The framework refers to PENMAN's components that lie outside of NIGEL. (Based on [177, 178])

grammar uses its systems and their associated choosers for making tactical decisions and invokes its realisers to apply the various linguistic rules.

The C4 categorisation is similar to C1, but its classes are slightly different due to the surge of data-driven approaches and the decline of canned text. In C4, the phrase-based and feature-based categories collapse into the grammar category. Statistical models refer to approaches that convert internal abstract representations (denoted s_i) to natural language text (denoted n) using the noisy-channel model $argmax_n p(s_i|n) = argmax_n p(n|s_i)p(s_i)$. The neural category refers to the use of neural networks, often deep, to generate text.

In the following sections, we will discuss the architectures used by surface realisers and our classification of existing realisers.

2.4.1 Classification of techniques

In this section, we will categorise the various existing realisation techniques similar to the C4 categorisation. However, unlike C4, we do not focus only on primary

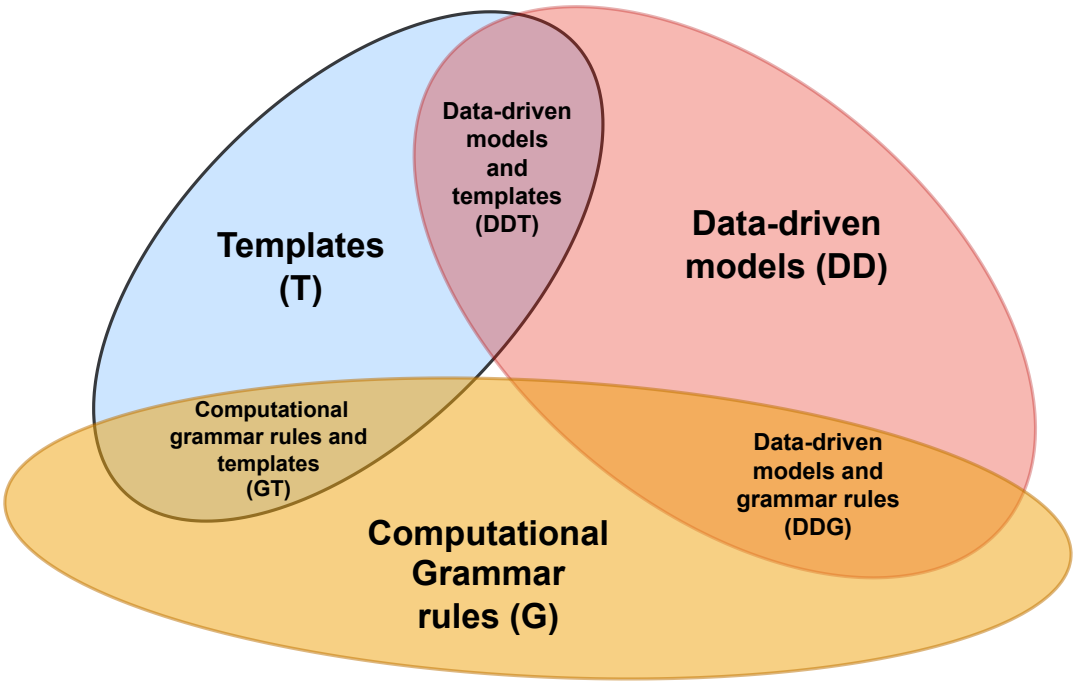


Figure 2.6: Existing surface realisation approaches. Abbreviations: T = templates, G = computational grammar, DD = data-driven models, GT = combined templates and computational grammar rules, DDT = combined templates and data-driven models, and DDG = combined grammars and data-driven models

categories. We also clarify the approaches that combine those methods, as illustrated in Figure 2.6.

2.4.1.1 Templates (T)

A traditional template is a sequence of unchanging words or word segments, at least one slot, and semantics for filling slots. An example of a template from a soccer summary generation system is shown in Figure 2.7. Templates are used in a variety of domains and implemented using different technologies (e.g., [249, 158]). They are suitable for languages, or domains, with little to no grammatical agreement. There are numerous template languages and associated template engines (e.g., Thymeleaf[2], Apache Velocity[3], Jinja[4]), and a large portion of them are designed for web page

[2] `https://www.thymeleaf.org/`
[3] `https://velocity.apache.org/`
[4] `https://palletsprojects.com/p/jinja/`

Original dutch: [**red player**] zou na [**minute**] minuten met een rode kaart het veld moeten verlaten nadat hij een overtreding beging.

Translated English: [**red player**] should leave the field after [**minute**] minutes with a red card after he committed a foul.

Figure 2.7: An example of template used to generate soccer text. Slots shown using square braces and bold highlighting. (Source: van der Lee, Krahmer, and Wubben [274])

templating but can be used for generating natural language text.

2.4.1.2 Computational grammar rules (G)

The second category of surface realisers uses hand-coded grammar rules. Systems of this nature come in two types: namely tactical generators and grammar engines [93]. Large scale realisers are responsible for tactical generation (i.e., tactical decisions and the application of grammar operations for linearising a sentence) while grammar engines are only responsible for the second task [94] and leave the first to the discretion of the engineer. We will briefly discuss the dated method of using large realisers and introduce existing grammar engines in this section. We discuss both types of CGR-based realisers to draw attention to the reason why extant NLG systems have opted for grammar engines instead of large realisers.

Tactical generators An early example of a tactical generator is the Penman generation system [179]. Its architecture is given in Figure 2.8. The Mann et al. [180] panel's recommendations influenced the design of the realiser. The system has the following modules and characteristics: a comprehensive natural language grammar, means for knowledge representation, a model of the text reader, and a model of discourse. In particular, it uses a comprehensive systemic functional grammar called NIGEL [177, 178]. It also uses the PENMAN upper model, a linguistic upper-ontology formalised in LOOM [17], and a domain model that is to be created by the engineer and aligned with the PENMAN upper model. The system also uses rhetorical structure theory for the creation of the discourse model [114]. It is superseded by Komet-Penman Multilingual (KPML), a multilingual generation environment [18].

Tactical generators such as GenDR [151] and AlethGen/GL [61] rely on Meaning-text Theory (MTT) as a foundation. MTT [122, 193] is a stratified feature structure-based

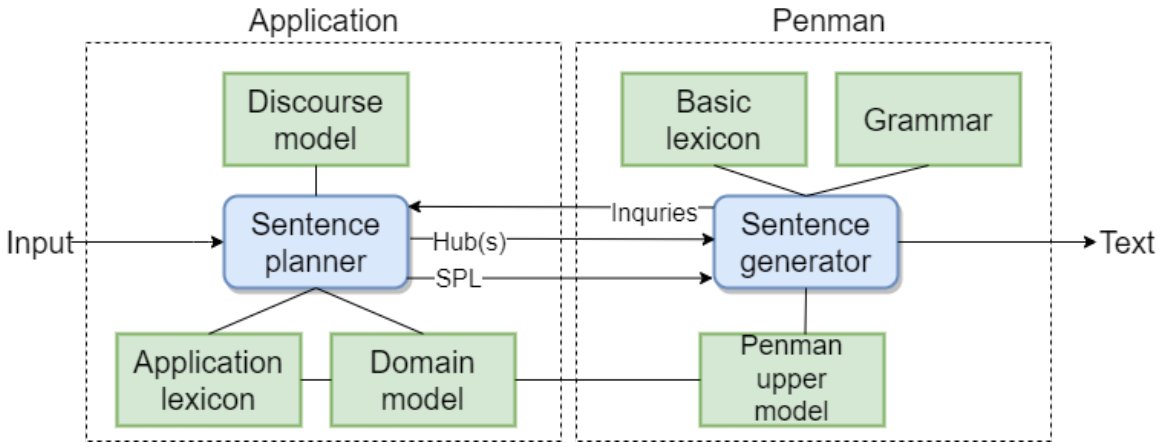


Figure 2.8: Architecture of the PENMAN reusable tactical generator. Abbreviation(s): SPL = Sentence planning language. (Based on [179, 114, 127])

framework that captures semantics, syntax, morphology and phonology. Similar to systemic functional grammar, MTT is reliant on feature structures. GenDR is responsible for tactical decisions via the semantic module and partial realisation tasks via a syntactic module. Overall, it takes in a semantic graph (SemR) and produces surface syntax representations (SSyntR). The realisation tasks are partial because it only transforms the deep syntactic representations (DSyntR) to SSyntRs and does not continue with the transformation to surface text. AlethGen/GL conducts both tactical decisions and realisation operations via grammar rules. In particular, it converts Events, custom structures produced by the sentence planner, to semantic representations (SemR). This conversion is essentially a transition from its additional stratum called the Events layer to MTT's semantic layer. It then uses the grammar rules to transform the SemRs, via the intermediate representation, to obtain surface text.

While the design decisions of such systems are sensible, the use of these realisers in contemporary NLG applications has gone down for practical reasons. These realisers exhibit control issues as a result of making tactical decisions their responsibility [94]. More precisely, such systems require a specialised input form and, as a consequence, they offer no direct way for control over how "phrases are built and combined, inflectional morphological operations, and linearisation" [94, p91]. In this work, we do not invest time building such large realisers for the languages in question as that is a risky endeavour, i.e., it is likely that NLG engineers would not adopt such a system due to the discussed control issues.

Realisation engines Most existing grammar engines use constituency-based grammar formalisms. These formalisms rely on constituency, the relation that exists between words when they tend to go together, and organise words in a hierarchical structure (e.g., *[S [NP This] [VP [V is] [NP a dog]]]* using bracket notation). The formalisms have two main components: the lexicons and elements that can be used to form the syntactic structure. When used in the context of NLG, the declarative lexicon component is paired with morphological categories and transformational rules. The SimpleNLG engines make use of such a grammar formalism for languages such as English [94], German [28, 36], French [277], Brazilian-Portuguese [74], Italian [186], Spanish [235], and Tibetan [146] using the Java, C#, and Python programming languages[5].

The components of these engines can be grouped into three categories: structural components of the text (i.e., list and document), lexical components (i.e., word and inflected word), and (3) phrasal components (e.g., coordinated phrases, phrase, etc.). The phrasal components are not all two-dimensional constituent structures since the canned text component is one-dimensional. We demonstrate how to use use SimpleNLG using the English template given in Figure 2.7, when captured using SimpleNLG we obtained the code listed in Listing 2.1 (explained afterwards):

Listing 2.1: Java source code for generating English soccer text using SimpleNLG

```
1       HashMap<String, String> infractions = new HashMap<>();
2       infractions.put("Matthijs de Ligt", "10");
3       infractions.put("Georginio Wijnaldum", "15");
4
5       for (String playerName : infractions.keySet()) {
6           Lexicon lexicon = Lexicon.getDefaultLexicon();
7           NLGFactory nlgFact = new NLGFactory(lexicon);
8
9           NPPhraseSpec slot1 = nlgFact.createNounPhrase(playerName);
10          VPPhraseSpec vpPhraseSpec = nlgFact.createVerbPhrase();
11          vpPhraseSpec.setFeature(Feature.MODAL, "should");
12          vpPhraseSpec.setVerb("leave");
13
14          NPPhraseSpec fldClause = nlgFact.createNounPhrase("the","field");
15          PPPhraseSpec minClause =
                nlgFact.createPrepositionPhrase("after",
                infractions.get(playerName)+" minutes with a red card");
16          PPPhraseSpec subClause =
                nlgFact.createPrepositionPhrase("after", "he committed a
                foul");
```

[5] https://github.com/simplenlg/simplenlg

```
17          SPhraseSpec rClause = nlgFact.createClause(fldClause, minClause,
                subClause);
18          SPhraseSpec sentenceStruct = nlgFact.createClause(slot1,
                vpPhraseSpec, rClause);
19
20          Realiser surfRealiser = new Realiser(lexicon);
21          System.out.println(
22                  surfRealiser.realise(sentenceStruct)
23          );
24      }
```

In the snippet, we begin by creating mock infraction data where we specify the offending player's name and the number of minutes they must leave the field (lines 1-3). We then retrieve SimpleNLG's default lexicon (line 6) to create the NLGFactory (line 7), a class for creating syntactic structures, and the realiser (line 20). The default lexicon's schema is given in Figure 2.9. We then create a phrase for the template's first slot using the player's name (line 9) and construct a verb phrase and noun phrase for the following fixed segments (lines 10-14). Following that, we construct the prepositional phrases where one of them includes the second slot (line 15-16). The final sentential structure is created using the smaller syntactic structures (line 18) and then linearised to produce the final sentence (line 22).

SimpleNLG influenced the jsReal(B), an engine which was created to support NLG for English and French in web development [200, 150]. JSRealB is a web-focused realiser and it is heavily influenced by SimpleNLG-EnFr [277]. Nonetheless, the engine differs from SimpleNLG with respect to the elements it supports, as can be seen in Table 2.7.

Another realisation engine that relies on a constituency-based grammar is Grammatical Framework (GF). Unlike SimpleNLG and jsRealB, GF supports a wide array of languages. At the time of writing, it supports 36 languages via its Resource Grammar Library (RGL). It also supports a wider range of structural, phrasal, and lexical components than SimpleNLG[6] and jsRealB[7] as can be observed from the framework's components in Table 2.7. The differences in lexical categories may not be significant as SimpleNLG has an inflected word type InflectedWordElement (via the NLGFactory's createInflectedWord method) and obtains its various forms through the use of features.

[6] https://github.com/simplenlg/simplenlg/tree/master/docs
[7] https://rali.iro.umontreal.ca/JSrealB/current/documentation/user.html?lang=en

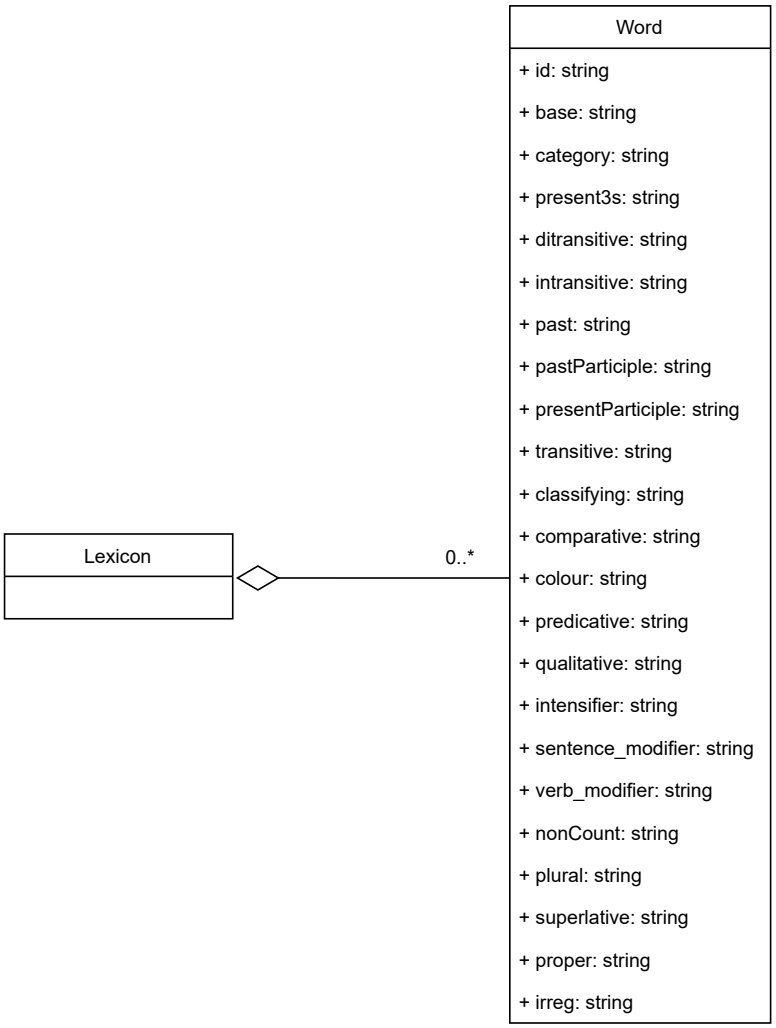


Figure 2.9: Schema used for SimpleNLG's default lexicon. A word has 23 properties that are associated with it. The meaning of the various properties can be found at `https://github.com/simplenlg`

While SimpleNLG, jsRealB, and GF are the largest constituent-based grammar engines, there also exist smaller constituency grammar rule sets built to support the generation of African languages (isiXhosa, isiZulu, and Runyankore in particular) [138, 45, 167]. These context-free grammars are limited to three lexical categories: nouns, verbs, and some prepositions. In the generators that rely on these CGRs, there has been no consideration for structural and phrasal components.

The only realisation engines that do not use a constituency-based grammar formalism, to the best of our knowledge, are RealPro [153], Lavoie et al.'s system [152], and OpenCCG[8]. Instead, the first two systems use MTT and take deep syntactic

[8]https://github.com/OpenCCG/openccg

Table 2.7: List of lexical, phrasal, and structural component types found SimpleNLG, GF's RGL, and jsRealB. The list of abbreviations, and examples for each, can be found in the GF RGL documentation [236, pg295-303], jsRealB documentation, and SimpleNLG Vers. 4 documentation

Engine	**Phrasal**	**Lexical**	**Structural**
SimpleNLG	Canned text, Coordinated Phr., and Phr. (N. Phr., V. Phr., Adj. Phr., Adv. Phr., Prep. Phr., and Sentence-like Phr.)	Word and Inflected Word	List and Document
GF RGL	IDet, Numeral, Card, Quant, Pron, Subj, PConj, A/A2, IQuant, Digits, AdN, Num, Det, Conj, AdV, V/V2/V3, IAdv, IP, CAdv, Ord, CN, Predet, AdA, PN, N/N2/N3	Phr., Utt, VP, S, Cl, RS, RCl, RP, VP-Slash, ClSlash, QCI, and QS	Text, ListNP, ListAdj, ListAp, and ListS
jsRealB	DT, NP, AP, AdvP, VP, PP, CP, S, and SP	N, A, Pro, D, V, Adv, P, C, NO, Q, and Raw/canned text	-

representations (DSyntR) as input to transform them to surface text via modularized grammar rules. The OpenCCG realiser expects input to be a hybrid logic dependency form [14] and uses a Combinatory Categorial Grammar (CCG) to transform the input to natural language (e.g., as can be seen in the OSU submission to the Surface Realization shared task held in 2011 [233]). All these engines conduct no tactical decisions as they move those responsibilities to the sentence planner.

Since the languages in question are low-resourced, reliance on only a grammar engine is not feasible. Instead, it must use a mixed-methods engine, similar to how CGRs are combined with templates by [138, 45] since grammar coverage is likely to be limited. The question of whether to rely on GF, Java, Python, or C# (à la SimpleNLG or its variants) comes to the fore when building such an engine. At first glance, GF seems like the better choice as one can achieve multilingualism through its RGL. However, such a feature is predicated on the creation of rules for each of the components listed in Table 2.7. This is not an easy feat for the languages in question due to their grammatical complexity and lack of up-to-date grammar literature. The existing

Table 2.8: An example of meaning presentation captured using production rules in PHARAOH++ (Source: [289])

$QUERY$	$\rightarrow answer(STATE)$
$STATE$	$\rightarrow state(STATE)$
$STATE$	$\rightarrow traverse\_1(RIVER)$
$RIVER$	$\rightarrow riverid(RIVERNAME)$
$RIVERNAME$	$\rightarrow ohio$

CGRs [258, 135, 167] do not cover nearly enough of those categories and they cannot be updated quickly. Consequently, at present, the choice between Java, Python, C# and GF is moot and can be revisited in the future when there is a mature set of CGRs with wide coverage.

2.4.1.3 Data-driven models (DD)

Most data-driven systems tend to use an integrated architecture hence have no dedicated surface realisation module. While such systems may have interesting differences (e.g., some are classification-based [182] while some are planning-based [12]), we will not analyse them here in this section due to their lack of a realisation component or something close it. The only exceptions are the systems submitted to the various (Multilingual) Surface Realisation Shared tasks.

The first wave of data-driven NLG systems relied on, mostly, phrase-based, SMT techniques. Popular examples are BAGEL [175] and Wong's [289] use of PHARAOH (and PHARAOH++) [142] for NLG. In BAGEL, Mairesse et al. [175] use a dynamic Bayesian network to transform semantic stacks to a sequence of phrases. A semantic stack is essentially a record that contains a variable name, its value, and a dialogue act type (e.g., inform(name(Zola))). Their model is also responsible for planning, in addition to realisation, since it needs to select relevant semantic stacks to be retained from the input (i.e., content selection). Wong [289] presents two SMT-based models: PHARAOH and PHARAOH++. PHARAOH is a log-linear reformulation of the noisy-channel model trained using minimum error rate and whose output 'translations' are obtained via beam search [239]. The difference between PHARAOH and PHARAOH++ is that the former treats the input meaning-representations as strings whereas the latter transforms the input into production rules before translation [289]. For instance, the input answer(state(traverse_1(riverid('ohio')))) is first converted into the production rules shown in Table 2.8. These types of realisation techniques have been superseded in popularity by Deep Learning (DL) models.

RNN-based (i.e., neural networks with recurrence) and transformer models are the most popular approaches for converting text plans into text (e.g., [51, 202, 57]). Both kinds of models use an encoder-decoder architecture, which is demonstrated in Figure 2.10. When the architecture is used in NLG, it can resemble the work done in MT where it is popular as well, in that it may take some sequence of words in order to produce another sequence of words. For instance, it has been used to generate text from a table (e.g., [161]), an image (e.g., [112]) for image captioning, or text (e.g., [247]) but for summarisation or expansion. With that said, most DL NLG models translate a sequence of text plan tokens to a sequence of natural language words. In the broader context, it must be noted that these systems can take in a variety of initial inputs.

The RNN-based models that abide by the encoder-decoder architecture operate by first converting the input text plan and output natural language text into some chosen vector representations, often dense, as shown in Figure 2.11. The vectorised text plan is fed into a Recurrent Neural Network (RNN) in order to generate some encoding of the input (denoted H in Figure 2.11). The encoded sentence plan and vectorised target sentence are then used as input into another recurrent neural network that is a language model (i.e., it predicts the next word given the previous word and vectorised text plan). When training, the predictions from the decoder are used to compute the loss and that is propagated backwards to the encoder (i.e., the model is trained end-to-end). Two special tokens are used to indicate the start and end of each sequence since the text plan and output sentence may have different lengths. These special tokens are denoted $\langle s \rangle$ and $\langle e \rangle$ in Figure 2.11).

After training, the decoder RNN is then used as a conditioned language model to generate text. The model is first fed with the $\langle s \rangle$ token, conditioned on the encoded sentence plan, in order to generate the first word. Until the decoder produces the $\langle e \rangle$ token, the previously predicted word is fed into the decoder, conditioned on the encoded sentence plan. The RNN-based models do not need to operate precisely as described as there are numerous techniques or design choices that are used to improve the quality of their output. For instance, one may choose to make use of

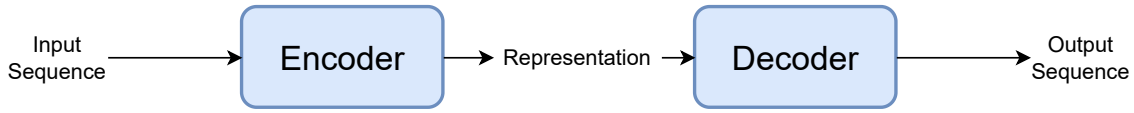


Figure 2.10: Representation of a sequence to sequence architecture

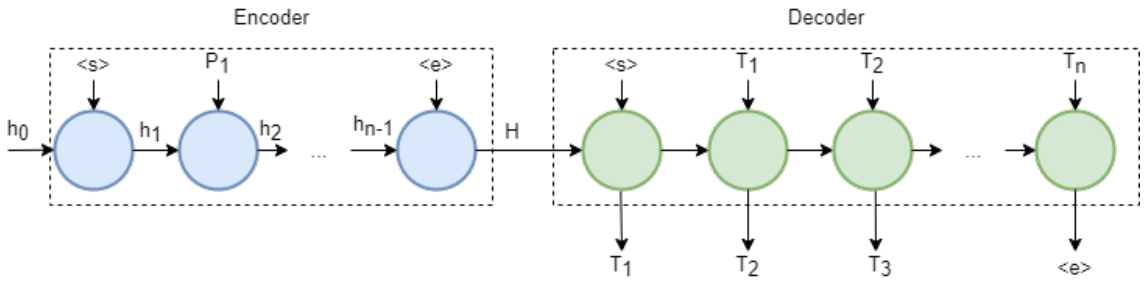


Figure 2.11: Representation of a sequence to sequence model that is recurrence-based when it is converting a sequence of plan tokens $(P_1, ..., P_n)$ into natural languages words $(T_1, ..., T_n)$.

greedy decoding instead of nucleus sampling, etc. [110], split the text generation into multiple and use a separate RNN-based model for each task [251], using a bidirectional RNN instead of unidirectional one, use a stacked recurrent neural network instead of a single layer, use Long-Term Short Memory or Gated recurrent units, etc. instead of a vanilla RNNs. Since the literature on deep learning models is rapidly growing and such a discussion is not pertinent to surface realisation, we do not attempt to offer any detailed discussion here and refer interested readers to Narayan and Gardent [206].

There are also transformer-based models that have been used to map the various inputs that are given to NLG systems to generate text. The popularity of such models is influenced by the effectiveness of attention [13], the possibility of speeding up transformer models [276] via parallel computation, and the prevalence of the pre-training strategy (e.g., [87]). Such NLG models use variants of the architecture given in Figure 2.12. When using the original transformer architecture, one creates vector embeddings for the input tokens, encodes positional information in them without significantly degrading the semantic information that is intended to be captured by the embedding, and then feeds them to the encoder. One then introduces weights that are dependent on the input/output pairs to scale the input vectors and feeds the result to a feed forward forwards neural network to generate some normalised representation. This aforementioned process can be executed multiple times. The encoder's representation of the NLG system's input is then fed to the decoder. A similar process is conducted by the decoder, however, its 'input' is the sequence of NLG system's output tokens. In addition, its first multi-head attention is masked.

We demonstrate the notion of masking by using a segment of the English template shown Figure 2.7 as a possible output, namely *Matthijs de Ligt should leave the field*,

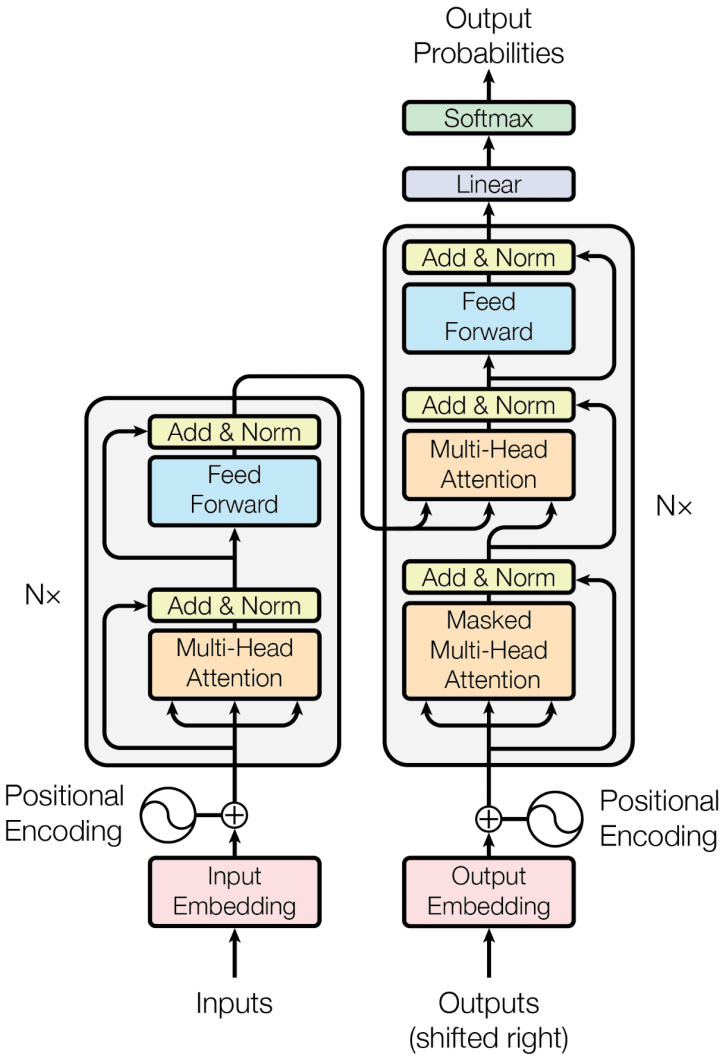


Figure 2.12: Representation of the original Transformer architecture (Source: Vaswani et al. [276]).

then the decoder considers the following values:

1. `Matthijs MSK MSK MSK MSK MSK`
2. `Matthijs de MSK MSK MSK MSK`
3. `Matthijs de Light MSK MSK MSK`
4. `Matthijs de Light should MSK MSK`
5. `Matthijs de Light should the MSK`
6. `Matthijs de Light should the field`

In the list above, we use the typewriter font to denote that the listed items are vector representations of the corresponding tokens and use the shorthand `MSK` to refer to the

representation of the mask. Moreover, since the decoder's goal is to generate the NLG system's output sequence, it produces the probabilities for the different outputs.

The number of systems that follow a neural approach has exploded over the years and they have explored variations of text-to-text tasks (e.g., style transfer [65, 265, 290, 155, 116], text summarisation [161, 1], question answering [4], and Multiple Choice Question (MCQ) distractor generation [125]). Even when their input type is limited to one category (e.g., data), we have also seen a lot of variations in the input forms. For instance, some data-to-text NLG systems take charts (e.g., [213]) while others accept meaning representations (e.g., [121]). Most neural NLG systems are not directly relevant to our work because they do not split the generation task into modules such that there is an identifiable surface realiser. Instead, their encoder creates a representation of the input and the decoder uses it to generate the final text.

Stated differently, most of the systems within this category that were created to have some notion of modularity (e.g., [231, 202, 269, 295]) use the dated architecture that splits the task into 'what to say' and 'how to say it' as described in [268, 255]. For instance, Puduppully, Dong, and Lapata's [231] approach is to create a text decoder instead of a surface realiser. They feed the text decoder content plans to produce text. Overall, their system takes tuples as input, encodes them into vectors using a multilayer perceptron, and weights them via an attention mechanism. They then use a bidirectional Long short-term memory (LSTM) to create vector representations of both the content plan and encoded input. The vectors are fed into the decoder, a recurrent neural network with LSTM units, and a copy mechanism, to generate the output. The only DL-based NLG system that uses the contemporary pipelined architecture, and therefore has an identifiable surface realiser in the usual sense, can be found in [51]. Most of its modules use the previously discussed methods, i.e, a RNN-based and transformer models, to create two versions of a model that is faithful to the tasks described in [68]. However, its surface realiser does not use a neural network and instead, it is rule based.

The only systems that deviated from that split are the submissions to the Surface Realisation Shared Tasks. The participating systems focused on the generation of text from Universal dependency structures [23, 197, 198, 199]. Most of those systems split surface realisation into two tasks, namely ordering and inflecting lemmas, and use the described models to achieve the aforementioned tasks. Specifically, the submissions can be categorised as follows:

- Training an SMT model to be used to generate the most likely ordering of 'grammatical relations' when given a tree and then iterate the tree in-order to generate the final text [103].
- Ordering the input tree's nodes using a classifier and inflect the lemma in each node using a seq2seq or SMT model (e.g., [232, 52]). Once the a tree whose child leafs also contain the inflected words, then one can traverse the tree in order to generate the text.
- Ordering the input tree's nodes using a seq2seq model and inflecting the lemmas found in each node using a hash table (e.g., [256])
- Ordering the nodes of the input tree and inflecting the lemmas in each node using two separate deep learning models, mostly which follow the seq2seq architecture (e.g., [16, 80, 292, 272, 111, 185, 251, 293]).
- Uses a single seq2seq model to learn the ordering and inflection of the information found in the input's nodes (e.g., [88, 85]).
- Finetunes and uses a pretrained model to generate text from a dependency tree (e.g., [257, 89])

Overall, all the data-driven approaches that have been discussed above are not suitable for languages or domains where there is no large parallel corpus from which to train; hence, they have not been employed for generating African languages. When we look at the sizes of the popular datasets used to build data-driven NLG models, as listed in Table 2.9, we see that most of them require large amounts of data. Since the first 5 datasets in the table have less than 1000 items in their training sets, it may seem reasonable to expect engineers and researchers working with the languages in question to consider creating similar sized datasets.

The expectation is not reasonable because closer scrutiny of those datasets shows that either the dataset is still large and the number shown in Table 2.9 is misleading or the resulting model trained from said data was outperformed by techniques that are not data-driven. For instance, Yermakov, Drago, and Ziletti's [291] work shows that they assume that BioLeaflets is useful when used with a large pre-trained model, hence they implicitly assume that there exists a larger dataset to train such a model. In addition, analysis of the results of SemEval-2017 Task 9 shows that the results produced by the data-driven systems were not good and leading the organisers to conclude that there is "a long way to go to reach fluency". In fact, the best performing system,

based on human evaluations, is not data-driven — it is the GF-based system called RIGOTRIO.

Now turning to the shared surface realisation tasks, it may appear that the SR18-ar and SR18-pt datasets can serve as motivation since in the first multilingual surface realisation shared task [197], the seq2seq model submitted by Ohio State University performed best for Arabic and a SMT system submitted by Tilburg University performed best for Portuguese. It may not be wise to invest a lot of money and effort solely on creating similar sized datasets in the hope that they produce similar automated metric scores. Especially since the languages in question have richer morphology than Arabic and Portuguese, automated metrics may not correlate with human evaluations [243], and the integration of a surface realiser that expects Universal Dependency structures as input as part of a larger system may not be trivial or possible in a variety of systems. Lastly, the numbers given in Table 2.9 pertaining to RotoWire [287] are misleading because they do not make it clear that the summaries are long documents.

It is unlikely that we are going to see the development of large datasets in NLG for these languages in the near future. Unlike MT, where authors have combined religious, government, and phrasebook parallel corpora (e.g., [183, 212]) to build models of varying quality, it is not sensible to build NLG models without domain-specific data. Since the languages are low resourced, creating such datasets, even for a single domain, is not sustainable as it may require a lot of financial investment since people do not volunteer without financial incentives [216]. At best, one can create a 'rule-based system' whose data is to be used to scaffold existing datasets. In such a semi-supervised setup, the 'rule-based system' captures human crafted knowledge and its main purpose is to assist the data-driven model. Even in such a set-up, if we want to create such a system in a principled way then the problem highlighted in Section 1.5 regarding our lack of a systematic and planned method for associating templates with CGRs needs to be addressed.

2.4.1.4 Templates and computational grammar rules (GT)

There are approaches that have been created over the years that combine templates with computational grammar rules in order to introduce some flexibility and linguistic complexity to templates. Examples of such approaches are patterns [136], syntax templates [273], YAG's template specification language [190], XtraGen's templates [263],

Table 2.9: The sizes of some popular corpora used by data-driven NLG systems for various languages and input types.

Name/Citation	Lang.	**Train**	**Dev**	**Test**
BioLeaflets [291]	English	1068	134	134
Bio-AMR v0.8 [184]	English	5,452	500	500
SR18-ar [197]	Arabic	6,016	897	676
Rotowire [287]	English	7,633	1,635	1,635
SR18-pt [197]	Portuguese	8,325	559	476
SR18-fi [197]	Finnish	12,030	1,336	1,525
SR18-nl [197]	Dutch	12,318	720	685
SR18-en [197]	English	12,375	1,978	2,061
SR18-it [197]	Italian	12,796	562	480
SR18-es [197]	Spanish	14,289	1,651	1,719
SR18-fr [197]	French	14,529	1,473	416
WebNLG [92]	English	25,298	-	-
LogicNLG [57]	English	28,450	4,260	4,305
LDC2016E25 [184]	English	36,521	1,368	1,371
E2E [211]	English	38,710	4,301	7,590
SR18-rs [197]	Russian	48,119	6,441	6,366
SR18-cz [197]	Czech	66,485	9,016	9,876
ToTTo [219]	English	120 761	7 700	7 700
Perez-Beltrachini and Lapata [220]	English	165,324	25,399	23,162
Chisholm, Radford, and Hachey [58]	English	401,742	50,017	50,030
WikiBio [154]	English	582,657	72,832	72,832

and the extant augmented template specification and engines such as RosaeNLG[9] or CoreNLG[10]. The aforementioned systems and their associated template extensions were created for various reasons:

- van Deemter, Theune, and Krahmer [273] created their syntax templates to support the generation of soccer reports in Dutch.
- Keet and Khumalo [136] created patterns, an ad hoc extension of template, to support ontology verbalisation in isiZulu
- YAG and XtraGen's template formalisms were introduced generic solutions to support a wide number of different domains
- CoreNLG and RosaeNLG are small augmented template engines intended for small NLG projects and are influenced by template processors for the web

[9] `https://rosaenlg.org`
[10] `https://societe-generale.github.io/core-nlg/`

All these extended templates keep the notion of slots and fixed words, elements they have inherited from ordinary templates. Some of them use rules to produce the fixed words instead of placing the words explicity. Unlike ordinary templates, they introduce additional rules that responsible for a variety of tasks. For instance, they may be used to attach annotations to words or have a higher function such as text planning). The complete list of components in each of the template types is given in Table 2.10.

In the Table 2.10, we introduce our own terminology for some constituents that are not named by the creators of each template type. The table shows us that Keet and Khumalo's [136] patterns have two kinds of slots: a sub-word slot and a template slot. A sub-word slot forms part of a word, its values are morphemes, and it differs from a regular slot in that it is not separated by a space from its surrounding constituents. The patterns also have two types of words: *fixed* words and *changing* words. The *changing* words have at least one sub-word slot and are obtained by the application of what we have termed the *Share affix* rule. This rule indicates that the affix to be inserted in a sub-word slot shares its noun class with the *changing* word's governor. This functionality allows the pattern to capture the concordial agreement found in isiZulu. Fixed words are regular natural language words. An example of a pattern taken from [134], for verbalising part-whole relations in isiZulu, is as follows:

$QCall_{nc_x,pl}\ \ W_{nc_x,pl}\ \ SC_{nc_x,pl} - CONJ - P_{nc_y}\ \ RC_{nc_y} - QC_{nc_y} - dwa$

In the pattern above, the last word (RC_{nc_y}-QC_{nc_y}-dwa) is a *changing* word and it has a sub-lexical slot for quantitative concords (QC_{nc_y}). A *share affix* rule exists between the first word, the universal quantifier (∀, 'for all'), and the second word, the entity that plays the whole ($W_{nc_x,pl}$) in the has-part relation, since the quantifier has a concord whose noun class depends on the noun that is the whole. In the pattern, this rule is denoted using the shared subscript (nc_x, pl).

YAG's template specification language only has traditional template slots and three content types: *Word, String, Punctuation.* The *String* type is a fixed string, *Word* is a changing word obtained via inflection, and *Punctuation* is a punctuation mark. It also has five rules that offer control over choosing the three kinds of content types. For instance, *If* chooses a content type if a condition is met, *Condition* checks multiple conditions and chooses a content type should a any condition be met, *Template* inserts the slot-value pairs and returns the ordered content values, *Concatenation* concatenates the result of content to another, and *Alternation* randomly selects content from a list thus avoids stale text. Figure 2.13 shows an example of a YAG template,

Table 2.10: List of components found in the extended templates. Abbreviations: Concatenation = concat.

	Control		Content	
Name / Citation	Annotation	Rule	Word	Morpheme
YAG	**Complete list undocumented**	Template, if, Condition, Concat.	Word, String, Punctuation	N/A
Keet and Khumalo	N/A	Share affix	Fixed Word, Changing Word	Fixed affix, Slot
van Deemter et al.	Topic, Condition	Attach, Combine	Fixed word, Category	N/A
XtraGen	Simple, Complex, Parameter	Selection	Inflect, String	N/A
CoreNLG	nlg_tags, no_interpret, post_eval	Concat., nlg_syn, nlg_num	TextVar, nlg_enum	N/A
RosaeNLG	nominal group dictionaries, recordSaid, hasSaid, deleteSaid, dumpHasSaid, getDumpHasSaid, getHasSaidCopy	value, *ref mixin, agreeAdj, eachz, syn_fact, synzsyn, syn, verb, subjectVerb, thirdPossession	String, itemzitem, dictionary	N/A

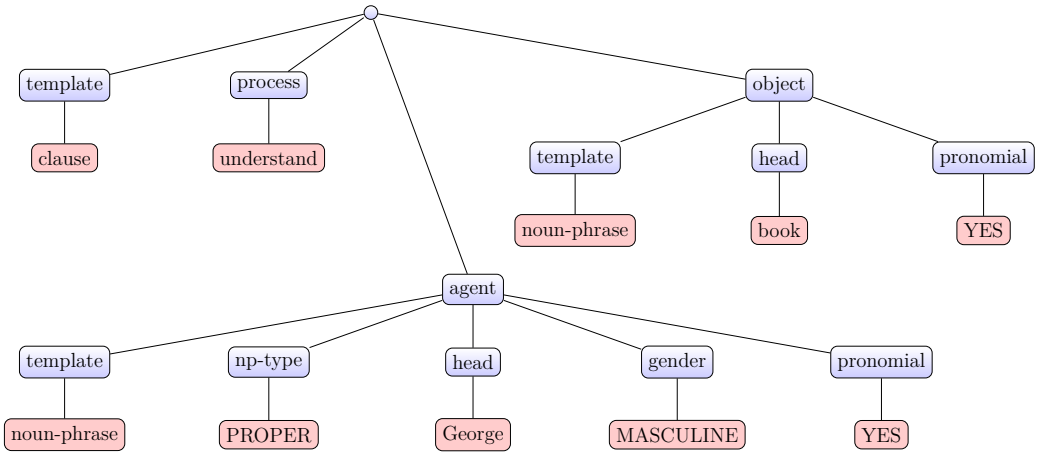


Figure 2.13: Tree representation of a YAG template (Source: [191])

represented as a tree, that was created to generate the text "He understands it". The template has two slots (i.e., agent and object) and they are filled with the values "George" and "book" respectively, in this instance. The agent's slot filler is annotated with a grammatical feature (e.g, masculine). It is also annotated with with "pronominal" so that the template generates the pronoun (i.e., the template is also responsible for referring expression generation). These slots and the fixed word (i.e., "understand") are organised using the ordering dictated by the 'clause' element.

XtraGen's extended templates only have a traditional template slot (Stenzhorn [263] calls it the Getter action). There are two main content types: a *string* and an *inflect*. The *string* type is a fixed string and the *inflect* type is a changing word produced via inflection. The extended template also has the *selection* rule that is responsible for using existing templates as building blocks in others. Lastly, XtraGen has annotation features that support the ability to specify when a template is suitable (via *Simple* and *Complex condition*) and add stylistic variation through the *Parameter* rule.

van Deemter, Theune, and Krahmer's syntax templates only have traditional template slots as well. The content types are categories for lexical and phrasal items (e.g. Nouns, Noun phrases) and fixed words. The categories are combined using what we call the *combine* rule and the lexical categories are attached to *fixed word*s via the *attach* rule. The lexicon, categories, and slots form what we call the syntax templates. These are syntax trees where some leaf nodes do not have lexical values but are slots. The syntax templates also have topic and condition annotations that can be used to specify the conditions in which the template is appropriate.

CoreNLG is a python-based resource and is one of the smallest augmented template

engines. Specifically, it only offers TextVar, a kind of text or canned text, for capturing the phrasal structure of the sentence to be generated. It does not have any dedicated Lexical element. Instead, it relies on python's in-built strings. The current version of CoreNLG only has the TextClass/Section and Document classes. The TextClass/Section classes both exist to order sentences and the difference between them is how the library permits access to certain functionality. Specifically, when using TextClass then NlgTools methods are accessed with python self reference. When using Section, the NlgTools methods are accessed using aliases[11]. For instance, one can use the add_tag method instead of nlg_tags. We demonstrate how to capture the English template listed in Figure 2.7 using CoreNLG in Listing 2.2 and explain it afterwards:

Listing 2.2: CoreNLG template for generating English soccer text using the CoreNLG engine

```
1  nlg = NlgTools()
2  infractions = {
3      "Matthijs de Ligt": 10,
4      "Georginio Wijnaldum": 15
5  }
6  for infract_key in infractions.keys():
7      text = TextVar(
8          infract_key,
9          "should leave the field after",
10         infractions[infract_key],
11         "minutes with a red card after he committed a foul."
12     )
13
14     text = nlg.write_text(text)
15     print (text)
```

The snippet shown in Listing 2.2 starts by initialising a NlgTools object; this allows CoreNLG to resolve language specific rules for contractions and typographical conventions. In the above snippet, we introduce mock data in lines 2-5, similar to lines 1-3 in the Java snippet presented in Section 2.4.1.2. We then iterate over the infraction data, order them using TextVar in lines 6-12. The write_text method in line 14 is then used to handle all post processing. Linguistic rules can be added to the above snippet. For instance, we can a synonym rule to obtain the template shown in Listing 2.3, explained afterwards.

[11] https://societe-generale.github.io/core-nlg/docs/framework/section/

Listing 2.3: CoreNLG template that includes a synonymy rule for generating English soccer text using the CoreNLG engine

```
1  nlg = NlgTools(lang='en')
2  infractions = {
3      "Matthijs de Ligt": 10,
4      "Georginio Wijnaldum": 15
5  }
6  for infract_key in infractions.keys():
7      text = TextVar(
8          infract_key,
9          "should leave the field after",
10         infractions[infract_key],
11         "minutes with a red card after",
12         nlg.nlg_syn("he committed", "their actions resulted in",
               mode="random"),
13         "a foul."
14     )
15
16     text = nlg.write_text(text)
17     print (text)
```

Listing 2.3 shows that the portion that of the template is in line 12. We have introduced synonymic phrases via the nlg_syn and the value to be used when linearising the template is chosen at random. The rules enables the generation of text that may either end with "...after he committed a foul" or "..after their actions results in a foul".

RosaeNLG differs from CoreNLG in that it includes a number of linguistic functions for enriching templates that do not exist as part and parcel of the templates but are external resources. The template structure can be captured via pug's[12] plain text, variables, and interpolation. It also offers verbs, possessives, and adjectives as lexical types via the language specific dictionaries. It also offers in-built linguistic functionality for capturing synonym via syn mixins[13], synz-syn structures, and syn_fct functions. At the time of writing, its lexicon covers America English, French, German, Italian, Spanish. Other functionality it offers pertains to control structures such as loops (e.g., eachz). For instance, we demonstrate how to capture the English template from Figure 2.7 and include a synonymy rule, similar to the CoreNLG template given in Listing 2.3. The resulting RosaeNLG template is shown in Listing 2.4.1.4 and we explain it afterwards.

[12] https://pugjs.org/

[13] A term that comes from pug to refer to functional-like reusable blocks of code

```
1  p
2       #{redplayername}
3       | should leave the field after
4       #{minutes}
5       | minutes with a red card after
6       |
7       synz {mode: 'once'}
8            syn
9                 | their actions resulted in
10           syn
11                | he committed
12      | a foul.
```

The snippet shown in Listing 2.4.1.4 generates texts that may either end with "...after he committed a foul" or "..after their actions results in a foul" similar to the CoreNLG snippet. In line 1 of Listing 2.4.1.4, we indicate that we are going to be generating plain text. This feature of RosaeNLG is carried over from the Pug templating engine. The above snippet also presents fixed segments in lines 3, 5, and 12. Its synonymic rule is captured via a *synz-syn* structure in lines 7-11.

This category of approaches (i.e., templates that use CGRs) is promising for isiXhosa and isiZulu, the languages that are the focus of this thesis. However, there is a need to consider what are the appropriate template components and how they must be combined. Currently, the discussed approaches treat the template as an isolated entity and their additions or extensions only exist alongside templates even though they are housed in a single file. In particular, a template is still a sequence of fixed words and slots. It may make use of control structures that are housed in the same file (e.g., synz-syn structures in the case of RosaeNLG). The manner in which the templates are paired with linguistic rules does not appear to be planned. There is a need to be deliberate regarding the type of pairing relations that are to be supported. Such deliberation might allow use to accurately assess each template type's novelty, verify the template's consistency, achieve interoperability, or be able to systematically examine each template's suitability for different languages.

2.4.1.5 Data-driven models and grammar rules (DDG)

When data-driven models are used together with grammar rules for text generation then the models are either used for ranking candidate outputs, probabilistic rule selection, morphological inflection, or word/lemma ordering.

More precisely, most methods that combine data-driven models and grammar rules use a framework with two modules: the first component is a base generator that

produces candidates and the second component is a selector/filter component. The main difference between the various methods is the choice of grammar formalism used by the candidate text generator and the method used for selecting the output from the candidates. For instance, White et al. [285] use a CCG and a chart generation algorithm to produce candidate sentences that are then ranked using an n-gram language model. Nakanishi et al. [205] use the Head-driven phrase structure grammar formalism and rank the candidate sentences using a bigram/syntax model. To the best of our knowledge, this generation framework was popularized in NLG by Knight and Hatzivassiloglou, Langkilde [140, 148].

In WASP^{-1} [289] and pCRU [20, 21], unlike the previous category of systems, the grammar is not used by a separate module to generate candidates that are then ranked by the statistical model. Instead, the grammar is used to build the statistical model thereby creating a bisecting architecture. The grammar can be created by hand (e.g., [20, 21]) or induced algorithmically (e.g., [289]). For WASP^{-1}, Wong uses a log-linear reformulation of the noisy channel model, and an Earley chart generator for decoding (see [289] for more details). The computational grammar is used for creating the WASP^{-1} translation model [289, p97]. For pCRU, Belz manually creates a Context Free Grammar (CFG) and then uses it to build a syntax-based language model via standard maximum likelihood estimation and add-1 smoothing. The model is then used to generate text via viterbi, greedy, or greedy roulette-wheel generation.

Another approach for combining data-driven methods and grammar rules can be seen in the English Treex/TectoMT surface realiser [83]. It makes use of the multilayer Functional Generative Description (FGD) linguistic theory and relies on hand-coded English grammar rules and a statistical model to transform each layer's representation to the next until one obtains a tree whose child nodes are well-formed English words. This realiser is similar to SimpleNLG in that the rules for combining words and phrases (i.e., the syntax rules) and the lexicon for defining categories and attributes are hand-coded. However, unlike SimpleNLG, the realiser uses its statistical model, a LIBLINEAR logistic regression classifier, for determining the morphological inflection of words. Kovács et al.'s [145] and Recski et al.'s [237] submissions to the Multilingual Surface Realisation Shared Tasks also follow this approach. However, they make use of Interpreted Regular Tree Grammar (IRTG) for capturing their grammar and use a seq2seq model, with attention, to inflect their lemmas.

The last category of DDG systems use the data-driven models for determing the correct order of the lemmas which are then fed into a grammar that is responsible for

inflecting them in order to obtain the final strings. For instance, Madsack et al.'s [164] submission to the 2018 Multilingual Surface Realisation Shared Task orders lemmas and their morphological features using (1) a seq2seq model with bidirectional Gated recurrent unit (GRU) units or (2) a two-layer neural network that expects two words as input. After ordering, they then use a proprietary grammar engine [281] to inflect of the lemmas to get a tree with well-ordered and inflected words.

The data-intensive nature of all these approaches disqualifies them, for the time-being or in the near future because of a lack of parallel corpus for the languages this thesis focuses on.

2.4.1.6 Data-driven models and templates (DDT)

Most methods that combine templates and data-driven approaches do so by making use of data-driven techniques to create candidate templates and then also using said techniques to select the best template, out of the candidates, when given some input for which the system is to generate text. Some only focus on a subset of the text generation tasks, e.g., the generation of only templates instead of the final text. For instance, Howald et al. [115] and Kondadadi et al. [143] extract templates from raw text using semi-automated methods that rely on the k-means clustering of text based on its semantic content. Once the templates are created, a suitable template for a given input is selected using a ranking algorithm in order to generate text. The process followed by that work to create the assets to be for text generation is shown in Figure 2.14. The authors start by splitting documents into sentences. They then identify named entities and semantic predicates using use the Boxer semantic analyser and custom and proprietary[14] name entity recognisers. Templates are then created by placing slots in the positions where the relevant named entities are found in the extracted documents. The templates are then grouped using k-means and the extracted semantic predicates to obtain template clusters that have similar communicative goals. When generating text, the algorithm first identifies the most relevant cluster and then uses an n-gram model to rank the cluster's templates.

Similarly, Angeli et al. [10] extract templates from raw text and these candidate templates are filtered using a data-driven model when generating text. They start by creating high-level categorisations of the input to be generated. For instance, in the case of the WeatherGov corpus, they identify inputs that pertain to the temperature, wind, sky cover, and rain. The granularity of these categories is based on the dataset.

1415

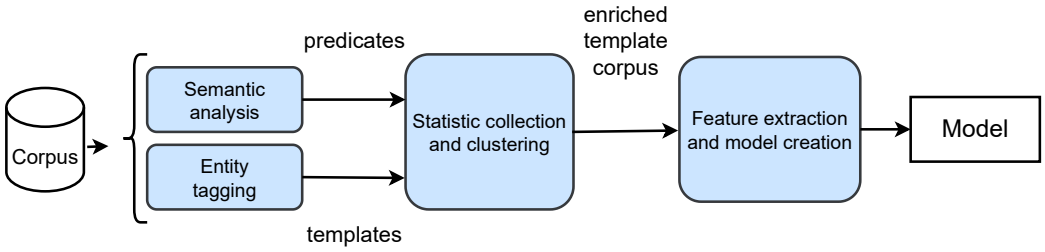


Figure 2.14: Representation of the process followed by Kondadadi et al. [143] to create their text generation assets (Adapted from [143])

Table 2.11: A collection of MRs that are aligned with text (Source: [10])

skyCover	temperature			
$mode = 50 - 75$		$time = 17 - 30$	$min = 44$	$mean = 49$
mostly cloudy,	with a	low around	45	.

For instance, they segment the wind category into windDir and gust. When learning templates, they align the category (what they refer to as record) and its associated values (what they call fields) with the text. For instance, when given the input $\{skyCover(mode = 50 - 75), temperature(time = 17 - 30, min = 44, mean = 49)\}$ and the text "mostly cloudy, with a low around 45." they produce the alignment shown in Table 2.11. One the input the input is aligned with the text, the Meaning Representation (MR)'s variables are used to replace the text. For instance, the phrase "low around" is replaced with the variable *time*. They then use this dataset to learn a log-linear model to be used to select the most likely template for each meaning representation[16].

A different approach is followed by Wiseman et al. [288] in their combination of templates and data-driven methods. Unlike Howald et al., Kondadadi et al., and Angeli et al. [115, 143, 10], they combine them in a joint model that learns/extracts templates and selects the appropriate template when generating text. Specifically, they make use of a encoder-decoder approach. However, their decoder is an extended hidden semi-Markov hidden model. After training, they use beam search to generate text in a "template-like way" [288]. The work done by Kale and Rastogi [123] is similar to this but they use a transformer to generate the templates.

Similar to the previous categories, these approaches are appropriate for the time-being or in the near future.

[16]Technically, the chain multiple log-linear models since the input contains multiple MRs

2.4.2 Summary

Of the discussed methods for surface realisation, templates are not appropriate for isiZulu and isiXhosa, unless the text to be generated is carefully designed to avoid grammatical agreement. For instance, instead generating the isiZulu text *amandla womoya azokwehla ngokuhamba kwesikhathi* 'wind speed will decrease over time' in a weather forecast NLG system, one could simple generate the text *amandla womoya: kuhla ekuhambeni kwesikathi* 'wind speed: decrease over time'. The underlined portion of the alternative text does not include a marker of what is being referred to hence is slightly awkward. Technically, this can be solved through the use of images next to the text in the case of a weather forecast NLG system. In general, crafting such text is a difficult feat that may result in awkward phrasing that goes against the communicative goals, especially where is not possible to include images as supporting material. The data-intensive approaches are not appropriate as a reusable solution, even when used in combination with grammar-rules or templates, because the languages in question are low-resourced. Unlike MT where some authors have relied on government, religious, or phrasebook corpora (e.g., [183, 212]), one cannot reasonably make a case for the use of datasets that are not domain-relevant in NLG. One could create a corpus by aggregating existing ones (à la AfroMT's isiZulu and isiXhosa corpora [240]) and enriching it with some input type (e.g., RDF triples, abstract meaning representations, etc.) to obtain a corpus of a similar size to the ones shown in Figure 2.9. Even if it were possible to train an NLG model on such a corpus, one cannot expect such a model to generate sensible text, say, in a restaurant use case since a significant portion of the existing corpora for isiZulu and isiXhosa is sourced from bibles, technical documents (e.g., ubuntu localisation files), and etc. Building NLG datasets from scratch for every new domain being tackled, to deploy a data-driven model, is not sustainable. Especially considering that existing work has shown that speakers of some of these languages do not participate in crowd-sourcing efforts unless there are financial incentives [216].

The only viable method reusable across different domains are templates that are combined with computational grammar rules. There are a number of existing programming languages, template specifications, and tools that can be used to obtain such augmented templates. For instance, one can make use of SimpleNLG or GF, or smaller tools such as CoreNLG and RosaeNLG, to generate English text (and support for other languages is possible). We now start by looking at the use of GF as a primary candidate; its RGL offers linguistic functions for 38 languages, hence it

seems sensible to use the language and its associated tools. However, its coverage for African languages is still minuscule [207, 229, 181] hence GF cannot be useful on its own. Similarly, while SimpleNLG was first built for English, there are variations for a number of European languages and Tibetan. It may seem like a second alternative is to build a Southern African variation of SimpleNLG. This is also not possible since building a grammar engine or resource grammar with wide coverage for the languages in question in order to obtain augmented templates is not trivial since the languages are under-resourced. A sensible strategy for the languages is to incrementally build a grammar engine by focusing on functions in the same fashion as Abed and Reiter [2]. In addition, the strategy would also make it possible to augment the limited grammar engine with linguistic rules that are not general enough to warrant their inclusion in the engine. Specifically, it would allow the combination of templates and domain-specific grammar rules in one entity as seen in 'patterns' [43, 135], the only approach that has been demonstrated to support African languages.

Patterns, a type of augmented templates, have wide domain applicability since their constituents should, theoretically, be able to support all domains. Practically, the existing ones [43, 135] are ad hoc, thus would have to be adjusted to be usable in another application domain that differs significantly from the ones they already support with respect to linguistic features. For instance, the original constituents in the isiZulu patterns [133] had to be amended with the possessive concord and locative affixes to support part-whole relations [134]. As a result, there is still a need to develop templates with sub-lexical components, with specific support for morphologically rich and agglutinating languages that can support any domain. Moreover, there is a gap to consider the use approaches that pair the artefacts while prioritising template scaffolding and CGR reuse instead of a specific domain's peculiarities. If the templates and grammar rules are viewed as separate entities, this opens opportunities for template specifications that support interoperability in template-based systems.

The CGRs used by these patterns are limited, as are the existing CGRs of other African languages (e.g., [15, 167, 207]); hence; the CGRs that are to be created must be used together with the templates. However, since the languages differ significantly from English (and other Indo-European languages) and their rules are to be used with templates since they are limited, it is inappropriate to adapt SimpleNLG and related tools for the languages. Instead, one should create a new grammar engine that exists in its own right and not constrained by the grammatical features found in an

engine that supports a language from a different family. For instance, if SimpleNLG's Application Programming Interface (API) were used, then phonological conditioning rules would not be prioritised despite their importance for word formation. The resulting engine could be adapted to GF, SimpleNLG, or related packages in the future, should the need arise, similar to Abed and Reiter's approach [2].

Chapter 3

Grammar-infused templates: combining template and grammar rules

In the previous chapter, we demonstrated that templates are not suitable for the languages in question when they are not combined with Computational Grammar Rules (CGRs). The combination of templates with grammar rules, leading to what is sometimes called augmented template systems, is promising as a solution for Nguni languages. However, the existing augmented templates systems have not been deliberate regarding how that pairing is done, hence may lead to systems that are hard to reuse and maintain.

The main aim of this chapter is to provide a formal definition of templates that are paired with CGRs. Henceforth, we shall call such templates *grammar-infused templates*. Such templates are key in broadening the languages supported within the Natural Language Generation (NLG) research community via the inclusion of Nguni languages.

Section 3.1 will demonstrate gaps in augmented templates systems, Section 3.2 will focus on understanding the possible ways of combining templates and CGRs, Section 3.4 presents the families of grammar-infused templates based on the supported relationships, Section 3.5 discusses how these pairings manifest themselves in the various existing grammar-infused templates, and Section 3.7 concludes with a demonstration the usefulness of the classified templates.

The work presented in this chapter was published in the International Journal of Metadata, Semantics and Ontologies [166]. It has been updated to improve its clarify

and readability. In particular, we added examples in Section 3.2.1, clarified how the features used in Table 3.2's categorisation were chosen, and reorganised Section 3.4 by separating the conditions that must be met for a set grammar-infused templates to belong to each class from the example of classifying a collection using the templates listed in Listing 3.6.

3.1 Gaps in existing augmented templates

To demonstrate the existing gaps with respect to the underlying models of augmented templates, we now turn to use the example of verbalising ontologies that capture knowledge about African animals.

In the previous chapters, we demonstrated that templates are not suitable for generating Nguni languages. In fact, this limitation stretches beyond Nguni languages and also affects other Niger-Congo B (NCB) languages. One of the limitations for templates is that they can produce texts with grammatical errors in cases when agreement is necessary.

In languages with limited grammatical complexity, unlike Nguni languages, one can solve this limitation through the introduction of certain phrase construction. In order to demonstrate their approach, consider the following template:

```
A [slot1] is a [slot2]
```

When used to verbalise the axiom $\mathsf{SubClassOf(Impala\ Animal)}$ we obtain the text "A impala is a animal" instead of the correct form "An impala is an animal". This problem can be resolved by using "a(n)" instead of either "a" or "an" (e.g., [118]). This amounts to hard coding all the possible values of some word, instead of relying on a CGR. This is not a practical solution for Nguni languages since agreement is mainly captured via concords and we have demonstrated in Section 2.1 that there numerous types of these concords, each category has numerous values, and listing them in a word without applying the necessary grammatical processes may be confusing. For instance, when using the aforementioned strategy for isiXhosa, the first two words of Byamugisha's [42, pg94] isiXhosa template for verbalising the axiom $\mathsf{Sheep} \sqsubseteq \forall\ \mathsf{eats.Grass}$ would be *(u/ba/i/li/a/si/zi/lu/bu/ku)onke iigusha* instead of *zonke iigusha* 'All sheep'.

People who are familiar with the augmented template-based systems discussed in Section 2.4.1.4 will recognise that the above issue can be solved by such systems, even if

one is not clear regarding their internal processing. When generating determiners using RosaeNLG, for instance, one has to use the value mixin which is integrated as part of the engine. As such, its processing is handled by RosaeNLG's ValueManager and the various language specific managers (e.g., PossessiveManager and GenderNumberManager). To demonstrate, consider the following template segment and the output in generates. We will explain it afterwards.

Template : #[+value('apple', det:'INDEFINITE', adj:'big')]
Output : a big apple.

In the above example, the template invokes the value mixin[1] and specifies the determiner key with a value of INDEFINITE (other options are DEFINITE, DEMONSTRATIVE, and POSSESSIVE). The above RosaeNLG template segment solves the issue regarding obtaining the indefinite determiner. This is achieved through the use of rules that are not found within the template. While such augmented template tools solve the aforementioned problem, the way they combine templates and rules is not planned.

To demonstrate the lack of planning, and its consequences, as observed in a tool like RosaeNLG, consider the following example: the tool can be used to capture synonymy in templates. However, unlike the case of the indefinite determiner, when capturing synonymy rules using the synz-syn structure[2], the linguistic rules are located in the serialised template and the engine is only responsible for using and applying the declared control features and structures. This can be observed in the following snippet that will be explained afterwards:

Listing 3.1: RosaeNLG template Snippet capturing synonymy

```
1 p
2   synz {mode: 'once'}
3     syn
4       | first alternative
5     syn
6       | second alternative
7     syn
8       | third alternative
```

When capturing synonymy in the manner described in Listing 3.1, one must note that RosaeNLG does not have a separate function, or any knowledge, that specifies that there exists a synonymic relationship between the *first*, *second*, and *third alternative* (lines 4, 6, and 8). That knowledge is only captured in the synz-syn structure found

[1] https://rosaenlg.org/rosaenlg/3.2.1/mixins$_$ref/value.html
[2] https://rosaenlg.org/rosaenlg/3.2.1/mixins$_$ref/synonyms.html

in the template. The engine's role is to select the appropriate linearisation and it will do so while avoiding to repeatedly select the same string since the *once* mode (line 2) is specified.

The decision to make the synonymy rules a part of the template vs. the approach of separating the rules from the templates as was seen for the determiners may suffice for RosaeNLG. In general, however, the problem is that there is no clear criteria for how that decision is made. In addition, there is no formal model we can use to properly distinguish between the two types of pairings. In the case of Nguni languages, the combination of templates with CGRs means that one has the benefit of using a simple technique such as templates and also deal with linguistic complexity when either the need or the desire for improved quality arises. In such a combination of resources, the nature of the pairing between templates and CGRs becomes relevant, especially for low-resourced languages where re-use of the scant CGRs is an imperative. It is then important to understand the possible ways of combining templates and CGRs and how they manifest themselves in the various existing systems that use templates in conjunction with CGRs.

3.2 Pairing relationships

In the following subsections we introduce the model, focusing on two relations, and formally define the two relations (namely, *embedding* and *attachment*). We also present the different kinds of grammar-infused template families that exist based on support for the various combinations of the two aforementioned relations (and their sub-relations), and illustrate how a collection of grammar-infused templates can be determined as belonging to each family.

3.2.1 Relationships

We propose a model that has two relationships, attachment and embedding, that exist between simple templates and CGRs, as depicted in Figure 3.1.

The attachment relation can be differentiated into two, *compulsory* and *partial attachment*, based on mode of participation. In order to formally define these relations, we first define simple templates and introduce our notation for CGRs. A simple template is define as follows:

Definition 3.2.1 (Simple template) *A bare, or traditional, template is a triple* $\langle L, f_o, f_s \rangle$*, where:*

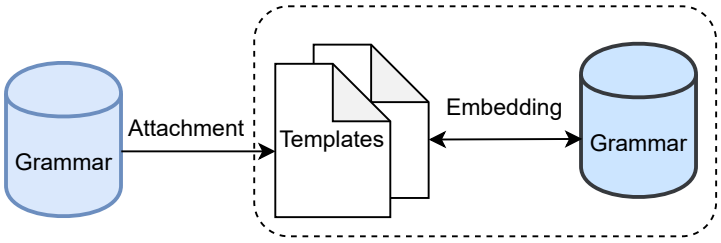


Figure 3.1: Grammar-infused templates where templates are paired with CGR sets through two kinds of relationships; attachment and embedding. The bidirectional arrow between the template and embedded CGRs denotes their co-dependent existence (embedding) whereas unidirectional arrow denotes independent existence (attachment).

- $L = W \cup S \cup A$ *is the template lexicon where* S *is the non-empty set of slot names,* W *is the non-empty set of words and meaningful word fragments from the natural language, and* A *is the set of the language's word separators/dividers;*
- $f_o : L \rightarrow L^n$ *is the template's lexicon ordering function whose output has at least one slot and one word hence the number of lexical items in the template is* $n \geq 2$*;*
- f_s *is a slot substitution function such that* $f_s : W \times S \rightarrow \{0, 1\}$ *outputs whether a word* w *is allowed to be used to fill a slot* s*. All words evaluating to true/1 for at least one* s *are assigned to some subset* $X_s \subset W$*.*

We now turn to give examples of template lexicon, ordering function, and slot substitution function. Continuing to use the template introduced at the beginning of this chapter `A [slot1] is a [slot2]` for illustrating the template lexicon: the first, third, and fourth 'words' are elements of W, the second and fifth 'words' are elements of S, and the five 'words' are separated by a single white-space which is an element of A. Continuing with our illustrative template `A [slot1] is a [slot2]` to demonstrate a template lexicon ordering function; we can create a function that takes the input { `a`, `[slot2]`, `A`, `[slot1]`, `is`, ws_1} where ws_1 denotes white-space of length 1, and then outputs the ordered list (`A`, ws_1, `[slot1]`, ws_1, `is`, ws_1, `a`, ws_1, `[slot2]`). Lastly, for our illustrative template, to demonstrate our notion of a slot substitution function: applications of the function $f(lexA, [slot1])$ and $f(lexB, [slot2])$ would return true/1 for any string lexA and lexB should they be lexicalisations of concepts A and B that participate in the relation SubClassOf(A B). Hence, the application $f(dog, [slot1])$ will return false/1 if the input is not an axiom where SubClassOf(dog B) for some concept B.

Simple templates have minimal and implicit linguistic rules to which they must adhere. Their rules pertain to the order of words through f_o, and other rules pertain to the semantics of each slot through f_s. More complex templates need to associate additional rules that exclude the rules for ordering the lexical items. Hence, in order to define such rules precisely, we need to define a collection of all lexical ordering functions for any given language.

Definition 3.2.2 (Ordering functions) *Given a language l and the set of all possible simple templates T, we define the set of all ordering functions for language l as $F_l = \bigcup_{i=1}^{\infty} \{f_{o_i}\}$ where $(X, f_{o_i}, f_s) \in T$ for some template lexicon X and slot substitution function f_s. This is the set of all the language's lexicon ordering functions as used by the simple templates.*

We then define the CGRs that can be associated with simple templates in order to form what we call grammar-infused templates, as follows:

Definition 3.2.3 (Associable grammar rules) *Let Γ be the set of all computational grammar rules of a natural language l, then Δ is the set of CGRs associable with simple templates in order to form grammar-infused templates and meets the following conditions:*

- $\Delta \neq \emptyset$;
- $\Delta \subseteq \Gamma \setminus F_l$.

For illustrative purposes, a member of the set defined above is the English noun pluralisation function $plrz$ that accepts a noun and its number to produce the plural form. The application $plrz(movie, plural)$ returns the string *movies* and $plrz(cherry)$ returns *cherries*. Other linguistic functions that belong to this set take different kinds of inputs, return strings which may be phrases, words or sub-word elements, and none of these functions are responsible for ordering lexical items. In the above definition, it suffices for our purposes to think of CGRs as functions that operate on words and their features (e.g., morphological features) to obtain words, phrases, and sentences. As such, it is possible that they break down a word or change some underlying morpheme to create another word. This definition is sufficiently generic to accommodate all the different extant formalisms that can be used to formalise a natural language's grammar. For instance, Meaning-text Theory (MTT) and Grammatical Framework (GF) use different internal structures. MTT is based on dependency-based structures and GF is based on constituency-based structures, but can be both viewed as a

collection of functions that operate on words and their features despite the differences in their underlying structures.

Simple templates and associable grammar rules can be associated together to form grammar-infused templates. We define the primary relations for combining the two assets as follows:

Definition 3.2.4 (Attachment and embedding) *Embedding and attachment are the relations denoted by the relations* embedded, attached *respectively. Given a set of simple templates T that uses the associable CGRs Δ of size $k > 0$ then:*

- *A CGR $g \in \Delta$ is embedded in a template $t \in T$, denoted* $\mathsf{embedded}(t, g)$*, through the relation* $\mathsf{embedded} = \{(t, g) \in T \times \Delta \mid t = \emptyset \iff g = id\}$ *where id denotes the identity function;*
- *A CGRs $g \subseteq \Delta$ is attached to a template $t \in T$, denoted* $\mathsf{attached}(t, g)$*, through the relation* $\mathsf{attached} = \{(t, g) \in T \times \Delta \mid t = \emptyset \not\Longrightarrow g = id\}$.

In the above definition, we equate the template, a triple, to the empty set and that is an overloading of notation. We use such notation because it is possible to have a set-theoretic treatment to tuples following Kuratowski [147, p42]. For instance, one can define an ordered triple in the following manner $\langle L, f_o, f_s\rangle \coloneqq \langle\langle L, f_o\rangle, f_s\rangle \coloneqq \{\{\{l\}, \{L, f_o\}\}\}, \{\{\{\{l\}, \{L, f_o\}\}\}, f_s\}\}$. To demonstrate the purpose of embedding relation, consider the template t = `A [slot1] is a [slot2]` (shorthand for the entire triple) and a function $synPhrase(template, phrase)$ for resolving the synonymic version of any/some phrase that is intrinsically linked with the template. If the two elements are related to each other via $embedded(t, synPhrase)$ then once t is deleted then the linguistic information encoded by $synPhrase$ also does not exist. Specifically, $synPhrase = id$; hence when it is applied to a word or phrase in the context of a template then returns the word/phrase unchanged (i.e., $synPhrase(t, phrase) = id(t, phrase) \coloneqq phrase$). We now turn to discuss the attachment relation introduced in Definition 3.2.4. The relation is similar to embedding in that it relates a template and an associable grammar rule. Unlike embedding, however, it captures the notion of independent existence. For instance, given the previously introduced illustrative template t and function $synPhrase$, then when the template t is deleted then it is not necessarily the case that the function $synPhrase$ loses all the linguistic information it encoded (i.e., $synPhrase \neq id$); hence the function $synPhrase$ continues to exist.

In definition 3.2.4, embedding specifies an existential dependence between a template and a CGR; hence, deletion of the template results in the loss of the grammar rule. When a CGR is lost in this manner, the function that encoded it becomes the identity function hence only returns the input it is given as opposed to applying any language specific rule. The embedding relation gives template creators the ability to scaffold simple templates. Attachment specifies an independent existence between a set of templates and a set of CGRs and there are two forms of the relations: partial and compulsory. The attachment relation supports the re-use of CGRs. The above definitions focus on a single template and CGR. We extend the relations to cover sets instead of singular templates and grammar rules. Specifically, in order to create the notions of partial and complete attachment of rules to a template, or set of templates, we create the following two relations:

Definition 3.2.5 (Partial and compulsory attachment) *Compulsory attachment and partial attachment are the relations denoted by* comp_attached part_attached*, respectively. Given a set of simple templates T then:*

- *The set of CGRs $G_1 \subseteq \Delta$ is compulsory attached to the set of templates T, denoted* $\mathsf{comp\_attached}(T, G_1)$*, through the relation* $\mathsf{comp\_attached} = \{(T, G_1) \in 2^T \times 2^\Delta \mid \forall t \in T\ \exists g \in G_1\ \mathsf{attached}(t, g)\ \wedge\ \forall h \in G_1\ \exists u \in T\ \mathsf{attached}(u, h)\}$*. ;*
- *The set of CGRs $G_1 \subseteq \Delta$ is partially attached to the set of templates $T_1 \subseteq T$, denoted* $\mathsf{part\_attached}(T, G_1)$*, through the relation* $\mathsf{part\_attached} = \{(T, G_1) \in 2^T \times 2^\Delta \mid \exists S_1, S_2 \subset T(T = S_1 \cup S_2 \wedge \mathsf{comp\_attached}\ (S_2, G_1) \wedge \neg \mathsf{attached}\ (S_1, G_1))\}$.

We continue to use our illustrative template to demonstrate the relations in Definition 3.2.5. Given our illustrative template and others related to it (i.e., belong to the set of of all simple templates, hence all belong to T) and a set of grammar rules in the same fashion as *synPhrase* that the belong to Δ, that are collectively referred to as G. Then if $comp\_attached(T, G)$ then it is not the case that every rule found in G might not exist when the templates in T are deleted. In other words, there is no existential relationship between the templates and rules. To demonstrate partial attachment relation, consider our illustrative template and related templates, collectively referred to as T and a collection of grammar rules related to, and possibly including *synPhrase*, collectively referred to as G. If $part\_attached(T, G)$ then if one where to delete every template T then a portion of the grammar rules in G will cease to exist since their life-cycle is tied to the templates or some of those templates did not make use of the rules.

In definition 3.2.5, we introduce specialisations of the the attachment relation in order to differentiate between CGRs used by all templates vs. a subset of templates. In particular, a CGR set is compulsorily attached if every template must use at least one rule from it and every rule is used by at least one template. A CGR set is partially attached to a set of templates if some templates do not have any attached CGRs.

3.3 Demonstration of relationships

We now turn to the illustration of the relations introduced in the previous section. Specifically, we do so by using the templates from three existing verbalisers designed by [100, 264, 72]. The Latvian verbaliser [100] illustrates embedding well: it makes use of CGRs that capture synonymy for some words and these rules are captured using regular expressions, for instance it uses the following template for class subsumption:

Listing 3.2: Latvian template for parsing and verbalising subsumption captured using GF

```
(Ikviens|Katrs) [slot1] ir [slot2]
```

The above template was created to parse or generate texts of the form *Ikviens profesors ir pasniedzējs* or *Katrs profesors ir pasniedzējs* 'Every professor is a teacher' to/from OWL axioms of the form SubClassOf(Professor Teacher). The first word in Listing 6.2.3.1 can have one of two variations (i.e., they are synonymic). The words' synonymic relationship is captured via GF's vertical bar operator. Using RosaeNLG's template specification, it takes the form shown in Listing 3.3.

Listing 3.3: Latvian template for verbalising subsumption captured using RosaeNLG

```
1 p
2       synz {mode: 'once'}
3             syn
4                   | Ikviens
5             syn
6                   | Katrs
7       | #{slot1}
8       | ir
9       | #{slot2}
```

In the RosaeNLG template given in Listing 3.3, the synonymic relationship between *Katrs* and *Ikviens* is captured via a synz-syn structure (lines 3-6). In both forms of the templates, the grammar rule capturing a synonymic relation is embedded in the template. In other words, if one were to delete the template file then this instance

of the grammar rule will no longer exist. The resolution of the values to be inserted into *slot1* (line 7) and slot 2 (line 9) would then be processed via JavaScript (with the help of a library that can parse OWL ontologies). The chosen lexicalisation of the concepts that participate in the SubClassOf relation would then be inserted into their corresponding slot by RosaeNLG's template engine. If templates would require the processing of certain linguistic rules then RosaeNLG (Version 3.2.1) would not suffice for Latvian, or any Nguni language, because their template processing engine only supports American English, French, German, Italian, and Spanish for certain grammar rules (e.g., verb conjugation).

Compulsory attachment can be observed in the BeInformed verbaliser [98, 72]: all its templates require syntax rules from GF's Resource Grammar Library (RGL) that persist even when the templates are deleted. The verbaliser was created for generating textual specifications of pre and post conditions for business processes that are captured using a specialised meta-model [98] inside the BeInformed platform[3]. For instance, consider the following business rule:

- An intake activity requires the submission of form

The above business rule is captured via the axiom requiresAvailable (Intake SubmissionForm) in Davis et al. [72]. For this rule, a grammar-infused template has been used to generate the English text "The intake may be completed, if the submission form is available". To demonstrate how the verbalisation is conducted via the function (i.e., grammar-infused template) and determine the relationship between the associated linguistic rules and underlying template, consider the following partial snippet in Lising 3.4.

Listing 3.4: Snippet of the GF template used to verbalise models in the BeInformed verbaliser

```
1  oper
2       intake_N = mkN "intake";
3       submission_N = mkN "submission";
4       form_N = mkN "form";
5       available_A = mkA "available";
6
7  fun
8       requires_available : Activity -> Artifact -> Fragment;
9       requires_available ac ar = mkFragm ac.subj ac.vp (mkS
10            (mkCl ar
11                (mkVP available_A)));
```

[3] https://www.beinformed.com/

The given grammar-infused template (lines 8 and 9) in Listing 3.4 takes an activity assigned to the variable ac (e.g., Intake) and the required artefact assigned to the variable ar (SubmissionForm). It uses GF's functions mkS, mkCl, and mkVP for ordering the items of the template (recall the ordering function in Definition 3.2.1). The grammar-infused template also makes use of the functions mkN and mkA, for instance, that apply the necessary inflections to nouns and adjectives. The mkA function makes it possible for the templates to produce "available" and "availability", depending on the context. The linguistic rules associated with those functions are not explicitly defined within the template. Instead, they are defined in GF's RGL; hence the deletion of the template will not result in the loss of the grammar for inflection.

Partial attachment can be seen in the SWAT verbaliser [264]. Most of its templates have attached syntax rules used for obtaining well-formed text in some slots, however, there are also templates that do not have such rules. For instance, the template used for generating text to specify that two individuals are different is given in Listing 3.5.

Listing 3.5: Prolog template used to verbaliser OWL axiom specifying that two individuals are not the same

```
1 s(differentIndividuals(Individual1,Individual2), Lexicon) -->
2   {traceRule(s:differentIndividuals/2:individualIndividual)},
3   name(Individual1, Lexicon),
4   [and],
5   name(Individual2, Lexicon),
6   [are], [different], [individuals].
```

The template in Listing 3.5 is taken from SWAT verbaliser source code[4]. For instance, when given the axiom differentFrom(LJJ_Wittgenstein LE_Boltzmann) where LJJ_Wittgenstein and LE_Boltzmann are instances of the concept *Person*, it can be used to generate a text like "Ludwig Wittgenstein and Ludwig Boltzmann are different individuals". The function name (lines 3 and 5) is used to retrieve the name of supported individuals from a predefined lexicon. There is no CGR used in this template, except for the implicit ordering of the various words. The *name* function that is used in Lines 3 and 5 do not capture any linguistic information.

The embedding and attachment relations give rise to different families of grammar-infused templates, which will be introduced in the next section.

[4] http://swat.open.ac.uk/tools/

3.4 Categories of grammar-infused templates

Given simple templates and CGRs, there are seven ways how the templates and CGRs can be combined to obtain grammar-infused templates, which are depicted graphically in Figure 3.2. These families arise based on a grammar-infused template's support for relations introduced in the previous section. They will be formally defined and illustrated in the remainder of this section.

In order to be able to easily identify these families, we need to first to formally define the concept of grammar-infused templates as follows:

Definition 3.4.1 (Grammar-infused template) *Given a simple template* $t_s = \langle L, f_o, f_s \rangle$ *that belongs to some set* T *and some associable grammar rules* A_p, A_c, *and* E. *We say that* $t_g = \langle t_s, A_p, A_c, E \rangle$ *is a grammar-infused template in the context of* T *if the following conditions are met:*

- $\mathsf{embedded}(t_s, g)\ \forall g \in E$;
- $\mathsf{part\_attached}(T, A_p)$
- $\mathsf{comp\_attached}(T, A_c)$

Given our formalisation of grammar-infused templates, we can be able to distinguish between templates created using various technologies. It may be tempting to assume that all grammar-infused templates, or augmented templates, created using RosaeNLG (or related technologies) treat the relationship between the simple templates and associable grammar rules in the same manner. Our formalisation can be used to identify the differences in how they combine the two artefacts. Specifically, a set of grammar-infused templates, T_g, can be classified into one of the seven families, based on characteristics of the following sets:

- the set of embedded CGRs (denoted with E)
- the set of partially attached CGRs (denoted with A_p)
- the set of compulsory attached CGRs (denoted with A_c)

Any collection of grammar-infused templates can be categorised as belonging to one of the seven families if it met the following conditions, respectively:

(1) *P* family: Its CGRs sets are such that $A_c = \emptyset$, $E = \emptyset$ and $A_p \neq \emptyset$ where $\emptyset$ denotes the empty set.
(2) *C* family: Its CGRs sets are such that $A_p = \emptyset$, $E = \emptyset$ and $A_c \neq \emptyset$.

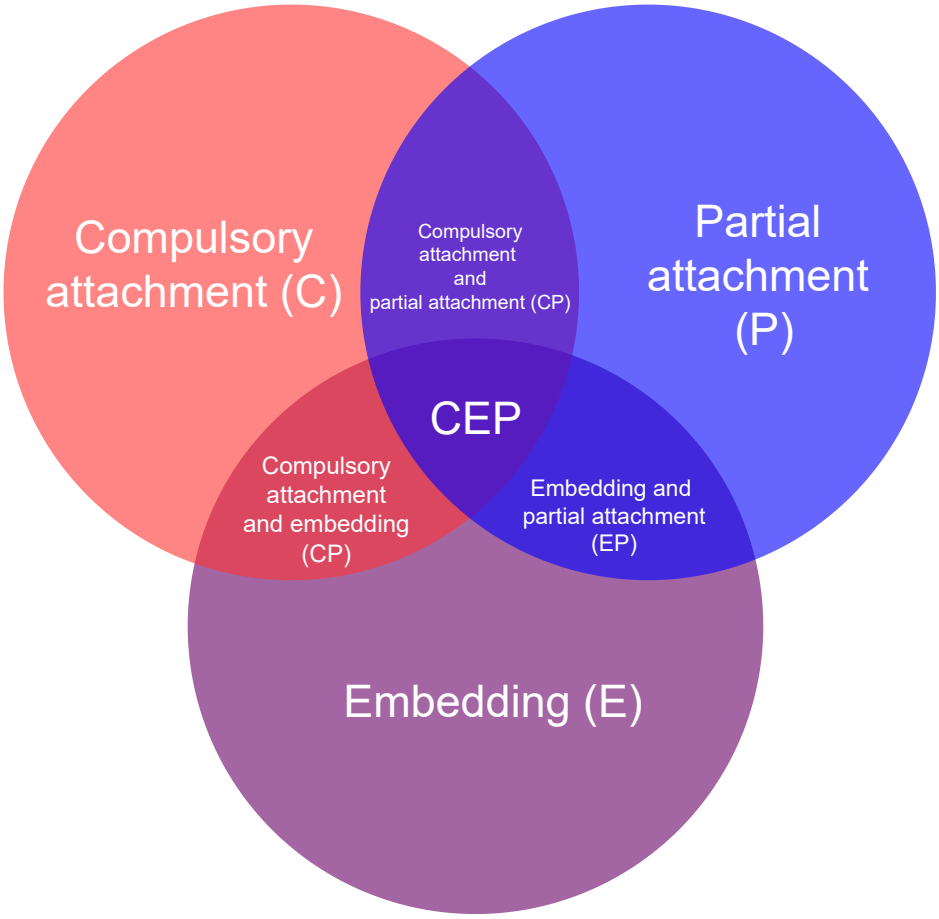


Figure 3.2: Seven different types of grammar-infused templates. CP, CE, EP, and CEP are combinations of the primary three relations of how CGRs can be related to templates.

(3) *E* family: Its CGRs sets are such that $A_c = \emptyset$, $A_p = \emptyset$ and $E \neq \emptyset$.
(4) *CP* family: Its CGRs sets are such that $A_c \neq \emptyset$ and $A_p \neq \emptyset$ and $E = \emptyset$.
(5) *CE* family: Its CGRs sets are such that $A_c \neq \emptyset$ and $E \neq \emptyset$ and $A_p = \emptyset$.
(6) *EP* family: Its CGRs sets are such that $A_p \neq \emptyset$ and $E \neq \emptyset$ and $A_c = \emptyset$.
(7) *CEP* family: Its CGRs sets are such that $A_c \neq \emptyset$, $A_p \neq \emptyset$ and $E \neq \emptyset$.

We will illustrate the classification criteria using the four templates in Listing 3.6 from a hypothetical bilingual verbaliser that generates English and Latvian text. These templates are formed by aggregated templates from Gruzitis, Nespore, and Saulite [100] and Stevens et al. [264]. We have chosen not create new templates for illustrative examples to avoid contrived examples.

Listing 3.6: Four grammar-infused templates encoded using a Definite Clause Grammar (DCG) in Prolog (adapted from [100, 264])

```
1 one(Term, Lexicon) -->
```

```
2       {Term =.. [sameIndividual|Individuals]},
3       {flatten(Individuals, Args)},
4       {IndividualList =.. [individualList|Args]},
5       [the], [following], [terms], [denote], [the], [same], [individual],
            [':'],
6       npList(and, singular, IndividualList, Lexicon).
7
8   two(functionalDataProperty(Properties), Lexicon) -->
9       {is_list(Properties), length(Properties, N), N>1},
10      [the], [following], [data], [properties], [are], [functional], [':'
            ],
11      verbList(singular, Properties, Lexicon).
12
13  three(sameIndividual(Individual1,Individual2), Lexicon) -->
14      name(Individual1, Lexicon),
15      [and],
16      name(Individual2, Lexicon),
17      [are], [the], [same], [individual].
18
19  four(subClassOf(Class1,Class2), Lexicon) -->
20      random_select([Ikviens|Katrs], uniform, Every),
21      name(Class1, Lexicon),
22      [ir],
23      name(Class2, Lexicon).
```

We recognise that our use of Prolog may be unexpected since it is not widely used in NLG, with the exception of ontology verbalisers. As such, prior to illustrating how to classify the above templates, we first explain what they do: the first template (lines 1-6) is used to verbalise multiple Web Ontology Language (OWL) axioms that specify that three or more individuals of some concept are the same. For instance, when given the axioms SameIndividual(A B and SameIndividual(A C) then it would generate texts of the form "the following terms denote the same individual: $\underline{\text{DT}_n \text{ nounA}_n, \text{ and DT}_m \text{ nounB}_m, \text{ and so on (3 items)}}$". We use the tag DT to refer to a determiner, in the fashion as the Penn Treebank project. The determiners' subscripts are used to denote agreement in number between them and their corresponding nouns. The underlined segment is generated by the function npList (line 6). Its primary function is to order the nouns associated with the individuals. However, in the given case, it also makes use of a function that inflects the determiner to ensure that it agrees with its corresponding noun.

The second template (lines 8-11) is used to verbalise axioms that declare functional data properties, i.,e. one "that can have only one (unique) value y for each instance" [19]. For instance, given the axioms FunctionalDataProperty(hasAge), FunctionalDataProperty(hasHeight), and FunctionalDataProperty(numberOfChildren), then it would generate "the following data properties are functional : [has age] and [has height],

and so on (3 items)". The underlined part is generated by the verbList function. The function is primarily responsible for ordering verb phrases. It also makes use of a linguistic rule for ensuring that the verb are singular (line 11) since functional data properties can have at most 1 value. The third template (lines 13-17) is similar to template one, but it applies for an axiom where there are only two individuals. For instance, when given the axiom SameIndividual(A B), it generates the text "A and B are the same individual". The name (lines 14 and 16) is used to resolve the lexical item for each individual. The fourth template verbalises OWL axioms of the form SubClassOf(Professor Teacher). It was first presented in Section 3.2 using GF and RosaeNLG's template syntax.

The four templates listed in Listing 3.6 make use of two types of rules: inflection and synonymy rules. In particular, templates *one* and *two* make use of inflection rules (via *npList* and *verbList*). These linguistic rules inflect verbs and determiners. Template *three* is a simple template, hence has no linguistic rules besides word ordering function. Template *four* has a rule that encodes synonymy for the adjective *every* (see line 20). The four templates would belong to each of the families if the following criteria were met:

1. P family: if the set of templates only had the templates numbered 1-3 in Listing 3.6, then the only linguistic rules that will exist in the templates would be the captured via npList and verbList. Since the linguistic rules would only be found in two of the three templates (i.e., templates one and two but not in template three) then that means $A_p \neq \emptyset$ and $A_c = \emptyset$, $E = \emptyset$, hence the collection would classified as belonging to this family.
2. C family: if the set of templates only included template *one* and *two*, as they are listed in Listing 3.6, then all the templates would have linguistic rules associated with them via npList and verbList. Since there no other linguistic rules, the collection would be classified as belonging to this family.
3. E family: if the set of templates only included template *four*, as listed in Listing 3.6, then the only linguistic rule in the templates would be an embedded synonymy rule. Consequently, the collection would be classified as belonging to this family.
4. CP family: if template *three* were removed from the set, template *four*'s synonym rules were redesigned to be attached, and template *four* also made use of attached linguistic rules (e.g., npList or verbList). In such a scenario, the npList/verbList rules would be attached (compulsory type) and the synonymy

rules would be partially attached, hence the collection would belong to this family.

5. CE family: starting with the set of templates given Listing 3.6; if template *one* were removed and template *four* were edited such that it also had attached linguistic rules then all templates would have attached rules and the only other rules would be embedded; hence, the collection would be classified as belonging to this family.
6. EP family: analysis of the templates given in Listing 3.6, then we see that the attached linguistic rules are found only in a proper subset of the templates and the synonymy rule is embedded in template *four*, hence the collection can be classified as belonging to this family.
7. CEP family: if the set of templates given in Listing 3.6 had additional CGRs that differ from the synonymy rules or the attached npList/verbList rules that were attached to all templates, then the set of templates would belong to this family since it already has partially attached syntax CGRs and the embedded synonym CGR.

We will show the applicability of this scheme by categorising existing systems and highlighting the tools' various differences in the next section.

3.5 Classification of grammar-infused templates

Since there are no published criteria that can be used for differentiating between different augmented templates, especially in relation to bootstrapping, then one is likely to use *ad hoc* criteria. For instance, one may choose a set of existing templates based on the programming language used by their corresponding existing tools and template specifications. Engineers who prefer to use the Python programming language may choose to use CoreNLG or Keet, Xakaza, and Khumalo's [138] templates since there are python tools that support them. The limitation of such criteria is that it does not consider notions of resource re-use and template scaffolding as being primary.

Our categorisation of grammar-infused templates resolves that. We illustrate the applicability of the categorisation of grammar-infused templates by classifying existing systems. The classification can be used when choosing an appropriate system from which to bootstrap when building new verbalisers. For the classification, we have chosen to categorise existing systems/tools based on their supported input type, the

type of grammar-infused template used, identifying the family to which the grammar-infused templates belong, the output languages generated by various systems, and determining whether these tools have some of aggregation or whether generation is achieved by simply slotting in values. We have chosen these features in order to understand the input types supported and their impact on the grammar-infused template design, they allow use to separate the grammar-infused template-types by language since techniques used for well-resourced languages may not be appropriate for a Less-Resourced Languages (LRLs) with morphologically-rich grammar, and also help us identify whether the grammar-infused templates have already been used in a system that supports aggregation since such techniques allow one to do away with redundancy. We recognise that some of the existing systems may not be reusable directly since their source code may not be publicly available. Nonetheless, we argue that our analysis may still be used to demonstrate that they are valid candidates and one can approach their original publishers to obtain the source code, where possible. In addition, the classification can be used to differentiate between future systems as well.

We collected a set of 54 ontology verbalisers and the tools used to create them. The tools and systems were then categorised into one of the five following main NLG approaches, where possible:

(1) *templates*: These are tools that make use of traditional templates only. These templates only have fixed words, slots, and values that can be inserted into the slots. In such systems, multiple templates can be used for variability when generating text from the a single input;
(2) *canned text*: Systems that make use of fixed text. Theses systems operate by concatenating short snippets of text together to form sentences;
(3) *grammar*: These tools use hand-crafted fully-fledged grammars for ordering and the inflection of all words — the engine has no fixing of words in a template-like fashion;
(4) *statistical methods*: These tools either make use of grammars that are induced from corpora or use small hand-crafted grammar rules combined with statistical models for re-ranking candidate realisations;
(5) *template + grammar*: These systems make use of grammar-infused templates. This is done either to obtain well-formed text in cases where traditional-templates are insufficient and full-fledged grammar engines are excessive, or to obtain some grammatical variability in the output text;

(6) *other*: Systems that do not fall into any of the previous categories (which may be because insufficient information was presented in the documentation).

The list of all the 54 systems and the categorisation of their realisation approaches are given in Table 3.1. We do not include CoreNLG and RosaeNLG in this analysis because they either do not have a publication describing them or there is no formal specification of their underlying language. In addition, the two libraries are relatively new and do not have a clear governance system; hence, it is possible that their developers would significantly modify them, possibly without explanation, such that our analysis no longer applies. In addition, it is not easy to trace the changes over time in tools that are not published. For instance, the current version of RosaeNLG that is available is 3.2.1, however, it is unclear how it differs from previous versions. Manually analysing the 93 versions (starting from v1.0.2 to v3.2.5) of RosaeNLG[5] to identify changes in its generation approach is deemed infeasible. Technically, the aforementioned issues may also affect published tools (to some extent). However, we argue that is a minuscule risk since those tools have the backing of research groups and/or private companies with a clear investment in the research community.

The table shows that 13 systems were categorised as using *templates*, 1 *canned text*, 10 *grammar*, 3 *statistical methods*, 16 *templates + grammar*, and 11 were categorised as belonging to the *other* class. In a similar fashion to our previous paper [168], systems that were not annotated with *template + grammar* were filtered out as out-of-scope, and the remaining 16 systems were then classified into their respective grammar-infused template family using the model introduced in Section 3.2. Their supported input and other features were also examined. The classification was done manually by analysing the grammar-infused templates found in research papers, technical reports, and PhD theses. The only systems for which source code was also analysed were the ones whose code was publicly released: the SWAT [264], multilingual museum [69], NaturalOWL [8], OWL-ACE [124], Latvian [100], and isiZulu [138] verbalisers. The classified tools are given in Table 3.2.

The table shows that 13 out of the 16 templates do not support the embedding of CGRs to conjugate words. It also shows that 8 of the 16 templates make use of syntax CGRs either for the entire sentence in the case of syntax templates or for a segment of the template in the case of partial syntax templates. Moreover, the various verbalisers and NLG systems expect different kinds of input and 82% of the

[5] `https://rosaenlg.org/rosaenlg/` [Accessed: 16-Oct-2022]
[6] This work also makes use of SimpleNLG's resources

Table 3.1: Categorisation of the 54 ontology verbalisers and associated tools based on the type of surface realisation.

Realisation method	**Reference to system/tool**
Canned text	Sadoun et al. [246]
Templates	Hewlett et al. [108], Jarrar, Keet, and Dongilli [118], Ang, Kanagasabai, and Baker [9], Al-Yahya [5], Ngonga Ngomo, Moussallem, and Bühmann [208], Power and Third [226], Lyudovyk and Weng [163], Liang, Stevens, and Rector [157], Liang, Stevens, and Rector [157], Halpin and Curland [104], Sanby, Todd, and Keet [249], Weal et al. [280], Casteleiro et al. [50]
Grammar	Amith et al. [7], Hielkema, Mellish, and Edwards [109], Aguado et al. [3], Lavoie and Rainbow [153], Bateman [18], Elhadad and Robin [86], Camilleri, Fuchs, and Kaljurand [47], Bouayad-Agha et al. [33], Bouayad-Agha et al. [32], Dongilli and Franconi [79]
Statistical methods	Cimiano et al. [60], Langkilde [148], White [283]
Templates+grammar	Grondelle and Gülpers, Davis et al. [98, 73], Stevens et al. [264], Androutsopoulos, Lampouras, and Galanis [8], Dannélls [69], Lim and Halpin [160], Davis et al. [72], Byamugisha, Keet, and DeRenzi [44], Keet, Xakaza, and Khumalo [138], Gruzitis, Nespore, and Saulite [100], McRoy, Channarukul, and Ali [190], Busemann [40], van Deemter, Theune, and Krahmer [273], Kaljurand and Fuchs, Kaljurand and Fuchs [124, 124], Wilcock [286], Stenzhorn [263]
Other	Mellish and Pan, Mellish and Pan [194, 195], Power [224], Coch [61], White and Caldwell [284], Piwek [221], Cojocaru and Trausan-Matu [62], Nguyen, Son, and Pontelli [209], Halpin and Wijbenga [105], Bouttaz et al. [35], Bontcheva [29], Bontcheva and Wilks [30]

Table 3.2: Classification and various features of grammar-infused templates for eleven verbalisers, three NLG systems, and two realisers that have support for grammar-infused templates (OWL = Web Ontology Language, XML = Extensible Markup Language, DCG = Definite Clause Grammar, GF = Grammatical Framework, Lang. = Languages, Embed. = Embedding, Conj. = Conjugation, Fam. = Family, Aggr. = Aggregation, and Enc. = encoding)

System/tool	**Fam.**	**Input**	**Lang.**	**Template enc.**	**Template type**	**Embed. verb conj. rules**	**Aggr.**
Verbalisers							
Davis et al. [98, 72]	C	GF	English, Dutch	GF	Syntax templates	No	No
Stevens et al. [264]	P	OWL	English	DCG in Prolog	Partial syntax template with number	No	Yes
Kaljurand and Fuchs [124]	C	OWL	English	DCG	Partial syntax template	No	No
Lim and Halpin [160]	P	-	Malay, Mandarin	C#	Pattern	No	No
Androutsopoulos, Lampouras, and Galanis [8]	EC	OWL	English, Greek	OWL	Sentence plan	Yes	Yes
Gruzitis, Nespore, and Saulite [100]	EC	OWL	Latvian	GF	Syntax templates with synonymy	No	No
Davis et al. [73]	EP	OWL	English	XML	Template[6]	No	No
Byamugisha, Keet, and DeRenzi [44]	EP	OWL	Runyankore	Java	Pattern	No	No
Keet, Xakaza, and Khumalo [138]	EP	OWL	IsiZulu	Python	Pattern	No	No
Dannélls [69]	C	OWL, GF	English, French, Italian, Finnish, Hebrew and Swedish	GF	Syntax templates	No	No
Hossain, Rajan, and Schwitter [112]	C	RuleML, JSON	English	DCG in Prolog	Partial syntax template	No	No
NLG systems							
Stenzhorn [263]	EP	XML	English, German, French, Italian, Russian, Bulgarian, Turkish	XML	Morphological template	No	No
van Deemter, Theune, and Krahmer [273]	EP	-	English, Dutch, German	-	Syntax template	No	-
Wilcock [286]	EP	XML	-	Extensible Stylesheet Language	Syntax template	No	Yes
Surface realisers							
McRoy, Channarukul, and Ali [190]	E	-	English	Template Specification Language	Template	Yes	Yes
Busemann [40]	E	Generation Interface Language	-	Template Generation Language	Rule	Yes	Yes

verbalisers that use grammar-infused templates are designed to generate text from ontologies. All the information in Table 3.2 can be used to make informed decisions when deciding to bootstrap a new system from existing templates. In particular, it can be used to identify any given language's existing systems and their corresponding grammar-infused templates. It can also be used to determine the suitability of each system's template type based on their supported input type and the characteristics of the realisation approach.

3.6 Demonstration of how to classify systems

In the previous section we classified a number of existing systems. We will illustrate how classification is done by using three verbalisers: Hossain, Rajan, and Schwitter, Keet, Xakaza, and Khumalo, Dannélls [112, 138, 69]. We chose to use these systems because they are recent, rely on different technologies, and generate texts that belong to different families.

The entity-relationship model verbaliser created by Hossain, Rajan, and Schwitter [112] currently supports only English as output, its templates are captured through a DCG in Prolog, and the templates have syntax trees that cover only a segment of the template. Listing 3.7 shows an example of the grammar-infused template in lines 1-3 that relies on some reusable rules. The template uses the np rule, which consequently relies on noun rule, for constructing noun phrase that is to be followed by the fixed phrase "is an entity type". When creating the noun phrase, the template specifies the whether their parsing or generating text (i.e., mode:M can either be mode:proc or mode:gen), specifies that the noun phrase should take the singular form via num:sg , and parses the entity (i.e., type:entity) that is going to be inserted in the slot whose position is marked by pos:subj.

Listing 3.7: Grammar-infused templates encoded using a DCG in Prolog for parsing a English sentence declaring an entity (Source: [112])

```
1  s([mode:M, type:entity, sem:L1-L2]) -->
2      np([mode:M, num:sg, type:entity, pos:subj, sem:L1-L2]),
3      [is, an, entity, type], ['.'].
4
5  np([mode:M, num:N, type:T, pos:P, sem:L1-L2]) -->
6      noun([mode:M, num:N, type:T, pos:P, sem:L1-L2]).
7
8  noun([mode:proc, num:N, type:entity, pos:P, sem:[L1|L2]-[[L0|L1]|L2]])
       -->
9      lexical_rule([cat:noun, num:N, type:entity, pos:P, sem:L0]).
10
```

```
11 noun([mode:gen, num:N, type:entity, pos:P, sem:[[L0|L1]|L2]-[L1|L2]])
       -->
12      {lexicon([cat:noun, wform:WForm, num:N, type:entity, pos:P, arg:_X,
           sem:L0])}, WForm.
```

In general, the templates verbalise entity, relationship, attribute, and constraint declarations by using linguistic rules captured through the functions vp, np and verb. The attached rules are responsible for constructing verb phrases, noun phrases, and conjugating verbs. Since all the templates use those attached rules and there are no other CGRs used hence they were classified as belonging to the C family of compulsory attachment in Figure 3.2 (i.e., all CGRs are attached).

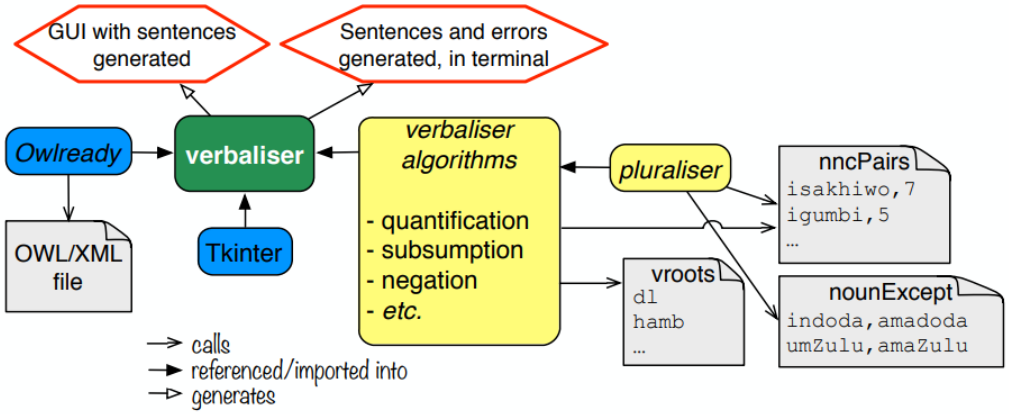


Figure 3.3: Architecture of the isiZulu ontology verbaliser that makes use of patterns (Source: [138])

Keet, Xakaza, and Khumalo [138] built a verbaliser prototype for verbalising OWL ontologies in isiZulu, a grammatically complex South African language. Since traditional templates were deemed inadequate for the language due to its complexity, they designed templates — which they call patterns — and accompanying algorithms for generating text. The verbaliser relied on Python and hand-crafted algorithms that capture the templates and the rest of its components are shown in Figure 3.3. In effect, the combination of templates and algorithms means that the templates make use of three categories of linguistic rules: morphological agreement, noun pluralisation, and verb conjugation. All the templates make use of the embedded morphological agreement rules. The noun pluralisation and verb conjugation rules are attached to a subset of the templates. This means that the templates only have embedded and partially attached linguistic rules hence the verbaliser belongs to the EP family in Figure 3.2.

Dannélls [69] built a verbaliser that generates descriptions of museum artworks. It makes use of GF to capture a painting ontology and the verbalisation templates. For instance, it can generate the English sentence "Big Garden was created in 1937" using the GF abstract and concrete functions in Listing 3.8. In the listing, the [69] relies on GF's separation of abstract and concrete syntax modules in order general notions about their grammar-infused template (via the abstract function specified in lines 1-3) and language specific linguistic information (via the concrete function specified in lines 5-8). The abstraction functions specifies that the template has two slots and they are reserved for a painting (i.e., Painting_Artwork) and the time period it was created (i.e., Painting_TimePeriod). The concrete function makes use of GF's internal function passiveVP to construct a passive verb phrase using the word "create" (referenced using the GF variable create_V2). It also makes use of the function mkAdv to construct an adverb that modifies a verb phrase. This adverb is constructed using a preposition (i.e., in_Prep) and the values insert into the second slot (i.e., o2). The passive verb phrase and adverb are then combined to form a verb phrase using mkVP. The constructed verb phrase and the value inserted into the first slot are then used to build the entire sentence using mkCl[7].

Listing 3.8: Declarations of abstract and concrete GF function that are used to verbaliser Paintility ontology (Source: [69])

```
1 fun Painting_hasCreationDate :
2  El Painting_Artwork
3     -> El Painting_TimePeriod -> Formula ;
4
5 lin Painting_hasCreationDate o1 o2 =
6     mkPolSentPast (S.mkCl o1 (S.mkVP
7     (S.passiveVP create_V2)
8     (S.mkAdv in_Prep o2))) ;
```

Overall, the verbaliser's templates use syntax rules from GF's Resource Grammar Library (e.g., mkVP in the template given in Listing 3.8). These rules allow the verbaliser to generate multilingual text. All the templates use the CGRs and the deletion of the templates does not result in the deletion of the rules since the RGL's rules are designed to be reusable. This means that the verbaliser belongs to the C family in Figure 3.2 since it only has compulsory attached rules.

[7]This function is for constructing declarative clauses

3.7 Utility

In this section, we demonstrate the usefulness of the notion of grammar-infused templates, its definition, and the classified systems. Specifically, we do so by demonstrating how one can pick appropriate grammar-infused templates when building a hypothetical children's game that is designed for learning about animals in isiZulu. The game can be thought of as a question generator from an ontology or conceptual model. In order for the game to be engaging, it would need to be accompanied by appropriate graphical content, however, we will not focus on that aspect since it is not relevant to our demonstration of utility. When deciding to build a system to generate such questions, the following steps, most of which were conceptualised by [25, p.1090], become pertinent:

1. Obtain description of domain from domain expert;
2. Create/choose provisional model;
3. Transform model into a mode that is understood by domain experts;
4. Request domain expert feedback;
5. Incorporate feedback;
6. If there are no conflicts, finalise the model; otherwise go to step 3.
7. Design templates for generating questions;
8. Generate question(s) on demand.

In this process, a provisional model or ontology can be built from scratch or extended from an existing one; hence the first step is optional. Whatever approach is taken, the created or reused artefact can only be determined as fit for purpose after a domain expert[8] approves of its contents. Since such experts may not have expertise with modelling languages then one can transform a model into a diagrammatic representation, natural language description or only use sample data when presenting it to them. Interpreting any of these layperson-friendly representations may also be time-consuming due to the cognitive load associated with a potentially large legend or glossary for diagrammatic representations or due to a possible difficulty in interpreting a potentially unclear automated textual report. As such, a structured approach that may be more efficient is the use of natural language generation to produce yes/no questions that probe the contents of the model, as advanced by [25], and can be incorporated in the activities already discussed (step 3 and 4).

[8]This descriptor includes both educational game designer and zoologists

Overall, this means that the engineer will have to create two sets of templates. The first collection would be designed for the domain expert(s) and another set would be designed for the final end user (i.e., players of the game). The generation of such yes/no questions is not novel, it can already be found in computer-assisted educational exercises that make use of ontologies (e.g., [53]). The current method of building template-based tools to generate these yes/no questions is to follow a two-step design process: selecting constructs to support and designing templates. In particular, one first decides on which constructs from a modelling language to support so as to limit the scope, depending on the expressiveness of the language. For instance, if an engineer chooses to start with the use of the knowledge captured in the African wildlife ontology [131] (and possibly extended for their game) then they would need to create templates for the following three OWL constructors, at the very least:

1. SubClassOf(C1 C2)
2. ObjectPropertyAssertion(OP I1 I2)
3. DisjointClasses(C1 C2)

For instance, when given the axiom SubClassOf(Lion Animal) and the isiZulu lexical items for the concepts *Lion* and *Animal* (*ibhubesi* and *isilwane*, respectively), one can generate the question *Ingaba azizilwane onke amabhubesi?* 'Is every lion an animal?'. The appropriateness of the generated question(s) largely depends on the templates and the audience they are designed for. More generally, the engineer needs to make decisions regarding the following:

1. *verbalisation*: The must decide on the choice of words to use in a template to express each construct, especially for morphologically rich languages (e.g. [100, 136]). This can be achieved by surveying the end-user's preference or it could be decided by the template designer. The definition of grammar-infused templates can be used to identify, together with the user's preference choices, to assess whether simple templates would suffice or linguistic rules must be added to the templates. For instance, if one determines that the template *Ingabe [C1] SC-dla [C2] kuphela?* is preferred to generate questions of the form *Ingabe amakati adla inyama kuphela?* "Do cats eat only meat?" and *Ingabe izinja zidla inyama kuphela?* "Do dogs eat only meat?" then the definitions may be useful to identify that the formation of *-dla* requires linguistic rules that may either be attached or embedded.

2. *slot design*: A decision also has to be made on the type of slots to be used in the templates. One decides on whether to use semantic variables that are *ad hoc* (e.g., Model-T's template [232]), linguistically motivated slots types (e.g., CLaRO's user-friendly templates [137]), or slot types from ontologies. We demonstrate the differences between the different slot types by extracting and modifying template segments from the systems described in [245, 232, 137]. The templates can be domain-specific names as can be seen in the template "Is [*name*] located in the [*area*] area, near [*near*]?" that can be used to describe restaurants. Here, [*name*] refers to the name of a restaurant, the term [*area*] refers to geographical area where the restaurant is located , and [*near*] refers to landmark that is close-by. Another approach would be to use syntax categories like in the template "Is there [*noun phrase*] for [*noun phrase*]?". This approach may result in templates that are easy to understand from the perspective of the end user, if they have some familiarity with basic linguistics[9], and the slot can accommodate a broader list of input values. At the same time, these kinds of template slots also introduce challenges with respect to managing which combination of noun phrases will yield gibberish. The last slot type uses notions that are directly tied to an ontology. The template "The [*CE1*] [*OPE*] [*CE2*]?" uses the notion of class expressions (CE) and object property expressions (OPE). This may make the mapping between templates and axiom types easier, but it also limits the re-usability of templates outside ontology verbalisation.
3. *bootstrapping*: In the event that verbalisation requires complex grammar-infused templates due to the nature of the intended output language and said language is under-resourced, one has to investigate the possibility of bootstrapping templates from a related domain or language to save time and effort (e.g., [42, p81-107]).

In the above process, our classification of existing grammar-infused templates in Table 3.2 can be used in choosing an existing form of grammar-infused templates prior to creating templates for another language. In particular, the table can be used, when bootstrapping, to identify the grammar-infused methods that exist for a related natural language and identifying the relationship between template and CGRs for a given type of grammar-infused templates. When building a question generator, it can be used to determine whether it is possible to embed CGRs for conjugating verbs and whether an existing kind of grammar-infused template already supports

[9]This can be useful in situations where the end-user is expected to be able to control the text being generated.

aggregation. The only other existing technique that has been used to measure the re-usability of grammar-infused templates is a string-edit based measure (see [42, p81-107]). However, such a measure does not pay special attention to the type of relationship that exists between the linguistic rules and templates. Our approach puts such a relationship at the fore and enables one to make the following choices:

1) should CGR reuse not be a priority and adequate verbalisation can be achieved using a few templates since there are only three functors, then templates that support the embedding of CGRs (E/CE/EP/CEP families) may suffice as either stand-alone or as a building block to a larger NLG system.
2) if the reusability of CGRs is a priority and the isiZulu noun pluralisation and verb conjugation rules [129, 136] are adequate, then one can make use of templates that belong to the C/P/CP/CE/EP/CEP families.

Until now, choosing an template language would be based on one's experience and exposure with existing work. For instance, experienced NLG engineers who work with English might choose the Template Generation Language [40] or Template Specification Language [190] while new NLG engineers might choose contemporary augmented template systems (e.g., RosaeNLG). Our Table 3.2 is useful because it shows that the only templates that support isiZulu, which have already been shown to be suitable for verbalising class subsumption [138], belong to the EP family, so then both possibilities can be supported. However, support for the first possibility would be limited should the other two constructors need to be verbalised via embedding other kinds of CGRs in addition to morphological agreement CGRs.

3.8 Discussion

In the previous section, we demonstrated the utility of our definition and classification of grammar-infused templates with special emphasis for isiZulu. We now turn to look at the grammar-infused templates in the broader sense in order to understand which ones support reusable linguistic rules across the different languages. In addition, we will discuss other differences we have found in the classified systems.

The classification shows that even though most grammar-infused templates (63%) do not currently support any form of aggregation (see last column in Table 3.2), most of them have detachable grammars, and therewith support grammar re-use in some form. It also shows that most grammar-infused templates (81%) do not embed CGRs to conjugate verbs. This limited support for CGR embedding also means that

most templates encourage grammar detachability through the attachment relation. Overall, this means that most grammar-infused templates are geared towards scenarios where there are limited resources thanks to their support or encouragement of grammar detachability.

The framework used to capture the templates and linguistic rules may also introduce differences even though the combined elements are of the same type. For instance, there are two main types of templates that use syntax CGRs: (i) syntax templates and (ii) partial syntax templates. A syntax template is a template with a syntax tree. A partial syntax template is similar to this, but the syntax tree covers only a subset of the sentence. In both kinds of syntax templates, the syntax CGRs are added to introduce a two-dimensional structure that is either used for multilingual template support or error correction. In particular, the GF-based templates use the syntax tree to obtain multilingualism through the RGL. This can be seen in the multilingual museum verbaliser [69] that supports English, French, Italian, Finnish, Hebrew and Swedish through the library. The Dutch soccer report generation system created by [273] uses the syntax trees to check whether referential and quantificational expressions are correct. This shows that that framework used to encode the templates introduces potential differences even when the template types are the same. In particular, while all the mentioned templates attach syntax CGRs, in partial or compulsory manner, the use of GF to encode templates offers multilingual support that is not available in syntax templates that do not use GF.

Lastly, we have learned from our analysis that there are a number of different general-purpose languages (and sometimes markup languages) used to encode the templates and the language used may impact a verbaliser's supported input languages. For instance, in our analysis of the multilingual systems (recall Table 3.2) we see that GF-based tools afford one with multilingual support through the RGL, but they do not generate text by working directly with ontology languages such as OWL. In such tools, the ontologies need to be transferred to GF (e.g., [11]) and this may be time-consuming for large ontologies. This shows that if one intends to introduce syntax CGRs and use GF to support multilingualism and grammar detachability, then that must be weighed against limiting the types langauges you can take as input.

3.9 Summary

In this chapter, we proposed a model that has two relationships for pairing templates and CGRs in Section 3.2, we used those relations to define seven possible families of grammar-infused templates that exist in Section 3.4, and we then identified the surface realisation methods used by 54 existing systems and classified all the systems that make use of grammar-infused templates in Section 3.5. We also demonstrated the utility of the classification in the context of picking an existing system, to be used as a foundation, when building a question generator for model validation in conceptual modelling.

We now turn to a discussion of the usefulness of the grammar-infused templates beyond choosing an existing generation method. Our definition of grammar-infused templates allows us to draw a clear line to separate the templates from the grammar rules. As a consequence, we observe that most existing templates do not capture sub-lexical constituents hence they would need to be extended, if they are to be used in combination with Context Free Grammars (CFGs), in order to support morphologically-rich languages. There are only a few existing grammar-infused templates that introduce sub-lexical constituents and of those systems, and only *patterns* [138, 44] support the generation of a Nguni language. Widespread adoption of grammar-infused templates, or especially ones with a variety of sub-lexical constituents, relies on the ability to compare and determine their suitability for each natural language. However, this is not possible at the moment since there is no shared conceptualisation of templates, a central component of grammar-infused templates, across languages that exhibit different types of morphology. In the following Chapter, we address this by creating a model that is formalised in OWL. The model could have been created without the formalisation that we presented in this chapter. However, we argue that our formalisation is beneficial as it allows us to be precise regarding what we mean by the term "grammar-infused template" and it clarifies how the various elements relate to each other.

Chapter 4

A task ontology for templates to support morphologically-rich languages

In Chapter 3, we differentiated between simple and grammar-infused templates in Definitions 3.2.1 and 3.4.1, respectively. Simple templates suffice for languages with simple grammar or in areas where one can make use of writing conventions. By languages with simple grammar, we are referring to ones with limited word-to-word agreement, and possibly disjunctive orthography, such as English. For isiXhosa and isiZulu, templates suffice in some areas, however, they need to be combined with Computational Grammar Rules (CGRs) to form grammar-infused templates so that they can be used in all cases. The purpose of this approach is to put greater emphasis on sub-word elements — something that has largely been ignored in surface realisation even though it has been explored in Machine Learning (ML)[1] as a way of dealing with out-of-vocabulary words.

In order for grammar-infused templates to be usable for generating text in isiXhosa and isiZulu, and allow for the reuse of the technologies, we need ways of representing both the simple templates and associable CGRs. When we scrutinise the two elements in this pairing, we see that capturing grammar rules is easy because there are numerous formalisms that are appropriate for Nguni languages (e.g., Grammatical Framework (GF)). As far the templates are concerned, the methods used by the systems listed in Table 3.2 are not sufficient as-is.

[1] e.g., `https://github.com/google/sentencepiece`

In order for the approaches listed in Table 3.2 to be appropriate for isiXhosa and isiZulu, they need to support the languages' complex morphology, have a precise definition of the various template concepts so as to support accurate assessment of each template type, and allow the use of computers to verify the templates' consistency or to achieve interoperability. We use the term "model" to refer to an explicit specification of the elements that can be used in templates and the relationships that exists between them. For instance, if the template language being used is Extensible Markup Language (XML) then the model is the XML Schema Definition (XSD) model. We do not use the term 'model' to refer to a ML model.

To demonstrate the limitations of the methods used by those systems, we shall start by focusing on the models of simple templates. An example of such a model can be seen in the work by Jarrar et al. [118]. The authors introduce an XSD specification for purposes of verbalising Object-Role Modeling (ORM) models. When templates are created from such a model in to use to use in situations where one needs to capture some grammatical agreement between words, developers often adopt writing conventions to overcome their limitations. As such, they are able to generate understandable texts for languages with simple grammar without using the "full machinery [of Natural Language Generation (NLG) that may also require] automated morphological analysis for each language" [118]. However, in order to support languages with complex morphology, we need to look to other languages for models that combine template and computational grammar rules, even if they are *ad hoc*.

Out of the sixteen verbalisers and NLG systems that rely on grammar-infused templates, as classified in the previous chapter's Table 3.2, only nine have support for languages other than English. Of those nine, only *patterns* [43, 135] have support for African languages. Patterns have been used for generating isiZulu [135] and Runyankore [43] text. They are extensions of simple templates that are able to capture some of the morphological agreement relations that exist between words. More precisely, patterns differ from simple templates with respect to three aspects they have introduced, namely: sub-lexical constituents such as concords, rules for specifying the source of the concord's value, and rules for forming words from an ordered sequence of affixes. These *ad hoc* templates are used in combination with limited CGRs in that they compensate for the simple template's grammatical limitations and, to an extent, the underlying simple template compensates for the inadequacies of the CGR(s). This approach has wide domain applicability, since the constituents of the simple template

should, theoretically, be able to support all domains. Practically, however, the existing patterns [43, 135] are ad hoc and of limited scope, thus cannot support language fragments that differ significantly from the ones already supported with respect to linguistic features. For instance, it was necessary to amend the original constituents in the isiZulu patterns (as first presented by [133]) with the possessive concord and locative affixes so as to support the verbalisation of part-whole relations [134]. Moreover, their linearisation algorithms are tightly-coupled with the application domain, the verbalisation of axioms.

These identified gaps demonstrate that there is still a need for a formalism or specification of simple templates, especially ones that take into account morphologically rich languages, so that we can support Nguni languages. Consequently, simple templates must be extended via the introduction of the sub-lexical components (à la [43, 135]). Such an extension will introduce added complexity with respect to the number of elements, relations, and constraints. As such, it is critical to have methods for verifying the consistency of the templates. In addition, since such templates are built to support low-resourced languages, the specification must support re-usability and interoperability across systems (i.e., systems for text generation, template management, discovery, etc.). In addition, this means that the specification should avoid being tightly-coupled with the CGRs or linearisation algorithms.

In this chapter, we develop a method of capturing simple templates that is appropriate for Nguni languages. More precisely, we develop the Task ontology for CNL Templates (ToCT), a task ontology for templates that contains knowledge so as to be able to deal with sub-lexical components. We use the term 'task ontology' in the same vein as Chavula and Keet's [55], to refer to a model that belongs to the layer where one specifies "language-specific scenario oriented knowledge to enhance specific computational tasks" [55]. We evaluate the task ontology through the use of competency questions. We show that the ontology can capture words that fall outside the competence of simple templates for languages with complex morphology. We also created a data-to-text and an knowledge-to-text system for Nguni languages whose templates are captured using ToCT. Their templates capture a variety of word types and sub-lexical features and they produce understandable and grammatically correct texts. The two NLG systems will be described later in Chapter 6.

In the following sections, we present the method used to develop ToCT, its content, validation, and demonstrate how to use it. Section 4.1 presents the method followed for developing the ontology, its content, formalisation in Web Ontology Language

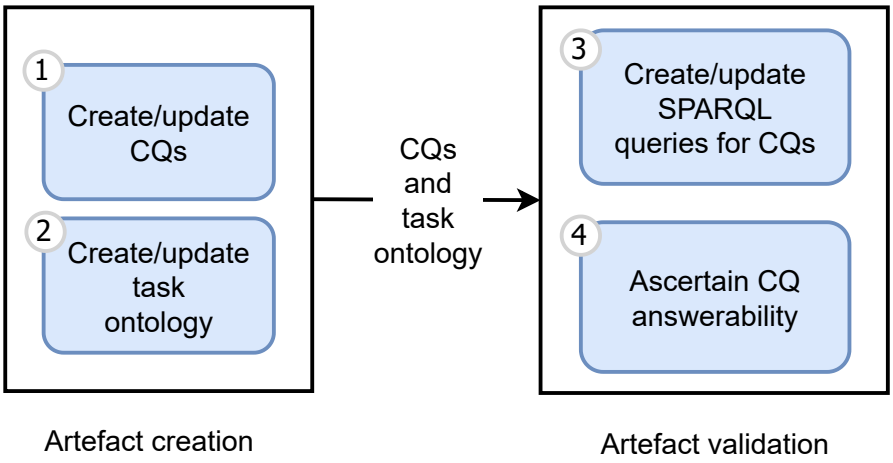


Figure 4.1: High-level representation of the process followed for each iteration in the creation of the task ontology. Abbreviations: Competency Question = CQ and SPARQL = SPARQL Protocol and RDF Query Language.

(OWL), and answerability of the Competency Questions (CQs). Section 4.3 presents a demonstration of how to use the ontology for two different languages.

The work presented in this chapter is based on work published at the *12th International Conference on Formal Ontology in Information Systems: Ontology Showcase*. Specifically, we have included additional examples of ToCT-captured templates and also described the additional iteration of ontology development.

4.1 Model development

We created the task ontology in three iterations, broadly following the main tasks in ontology development methodologies as summarised in [254]. Specifically, we carried out four main activities: competency question (CQ) creation, knowledge gathering, ontology creation and formalisation, and validation. A high-level visualisation of each iteration is given in Figure 4.1.

The first iteration went through the entire development process and focused on knowledge acquisition and support for isiXhosa, and the second iteration focused on improvements for applicability and extensions to also suffice for both isiXhosa and isiZulu. The third iteration focused on cleaning up with respect to the object property names used and capturing linguistic annotations.

In the rest of this section, we will use the process shown in Figure 4.1 as a guide to discuss the creation of the task ontology. Specifically, we will first discuss the final

competency questions, the creation and formalisation of the task ontology, and how we ascertained that the task ontology.

4.1.1 Competency questions

We created the following competency questions, as of the initial iteration, to indicate the scope of the model. The design of the above CQs is based on our domain knowledge regarding English-oriented templates, knowledge of isiXhosa as an L1 speaker[2], and knowledge of isiZulu as an L2 speaker and researcher. The goal was to capture the template features as they are found in existing English-oriented templates and also the features needed to address the gaps so that they are appropriate for Nguni languages. These gaps are addressed via the inclusion of sub-lexical constituents and specifying word-to-word grammar dependencies. The resulting CQs are as follows and we will explain them afterwards:

Q1: How many fixed phrasal and lexical segments does [template] have?
Q2: How many words that depend on others does [template] have?
Q3: Which properties may result in a change of form between [word1] and [word2] where there exists a dependency?
Q4: If there is a dependency between [word] and [word], which word is the governor?
Q5: Does [word] have a constant base form?
Q6: Which grammar rule will be activated when forming [word] if its dependent on another word?
Q7: Can [word] ever be placed in [slot]?
Q8: Can the word ordering in [template] exist without the template?
Q9: Can the word portion's ordering in [word] exist without the word?
Q10: Which words use [grammar rule]?
Q11: Is [template] grammar-infused?
Q12: How many slots does [template] have?
Q13: How many fixed segments have more than one word?
Q14: For each [word], in what order are its associated grammar rules applied, if at all?

In the CQs above, we use the square brackets are used to denote variables. For instance, Q12 can take the forms 'How many slots does template2 have?' and 'How many slots does tml9 have?' (among many) where the *template2* and *tm19* are possible values for the variable *[template]*. In the templates, we also use terminology that was

[2]Elsewhere in the literature, the term native or first language speaker is used.

introduced in Chapter 3. For instance, we use the term "slot" in the usual sense and "a change of form" to refer to the usual change of a word's surface form due to the application of linguistic rules.

The 14 CQs are translated into SPARQL Protocol and RDF Query Language (SPARQL) queries and the translatable questions were used to query the formalised model. The CQs and their corresponding SPARQL queries are listed in Appendix A. The answerability of the CQs will be revisited in Section 4.1.4.

4.1.2 Ontology creation

We sought to reuse existing ontologies to aid in the conceptualisation and formalisation stage, as it is best practice. We reused the Collections Ontology (CO) [59] and the Model for Language Annotation (MoLA) [95] for modelling a template's sequence of items and a rich set of of language features about the languages supported by a templates, respectively. Specifically, we created a module from the CO (omitting the not needed Disjoint(Collection Item) axiom) for ordered and non-unique associations. We chose to use MoLA since it captures a broad set of linguistic information (cf. its alternatives). Since it captures information about dialects and regions, that information may be useful when reusing templates for a new audience. The collections ontology was chosen since it is the only OWL ontology, to the best of our knowledge, that can be used to capture generic knowledge collections (see Walls et al.'s [279] Biological Collections Ontology) and infer knowledge about them even when there is "incomplete information" [59].

Since the ontology to be developed concerns linguistic concepts, we had considered using existing ontology lexicalisation models as a base. However, they do not have a large number of the required features; hence, adapting them means spinning off a variant in addition to adopting their idiosyncrasies. Notably, ontolex-lemon and GOLD and related artefacts are not usable off-the-shelf [54, 55] for the languages in question. The only model built for such languages, i.e, Bantu Language Model (BLM) [31], requires one create all forms of a lexical entry instead of creating them 'on the fly' when they are used in a template. It also has no notion of a template and provides no means for specifying dependencies between words. We ended up not using those resources because of the stated reasons.

Concerning knowledge gathering for the iterations, we relied on primary sources such as grammar textbooks, that describe morphology and grammar. To support isiXhosa

and isiZulu, we relied on Meeussen, Katamba, Bourquin [128, 192, 34] as primary sources for the different kinds of concords and parts of speech that exist in the languages. To determine how these concords are used together with other morphemes to form words, we relied on [214, 189, 81, 82] for isiXhosa and [48, 248, 49, 78, 64] for isiZulu.

The first iteration's conceptualisation included linguistic annotation concepts (i.e., syntactical and morphosyntactical properties). During the iteration, the goal was to create a modular model, à la *lemon* [187], in order to to avoid having to use every aspect of ToCT, the ontology presented in this chapter, even when it is not needed. The second iteration revealed differences in concord types between isiZulu and isiXhosa. Those differences, when coupled with the possibility of needing to update the linguistic categories when supporting a new domain [133], motivated the need to create separate and language-specific conceptual models. Eventually, in iteration 3, we created a separate task ontology for linguistic annotations and corresponding language-specific axiomitisations, following Chavula and Keet's framework [55]. This approach means that language-specific knowledge about linguistic properties, cf. template-specific knowledge, can be kept separate. This avoids limiting the template ontology's suitability to only isiXhosa. The development and content of the isiXhosa and isiZulu linguistic annotation ontologies will be revisited in Section 4.2.

The ontology is shown informally in Fig. 4.2, which was then formalised in OWL, resulting in a so-called 'task ontology', called ToCT, intended for the specific task of systematic, reusable, and interoperable template specification.

4.1.3 The ontology's content

In the ontology, a template is a sequence of ordered words, slots, spaces, and punctuation. A `Word` has the usual meaning. Similarly, The `Space` and `Punctuation` concepts have their usual linguistic meaning.

Words There are two kinds of words in the model; fixed words and changing words. The two types are captured via the `Unimorphic word` and `Polymorphic word` concept. The meaning of each of those two concepts is as follows:

- `Unimorphic word`: these are signs with a single written representation. For instance, the first word in the short English template "the [animal]" can only have that written representation, irrespective of what value is inserted into the slot.

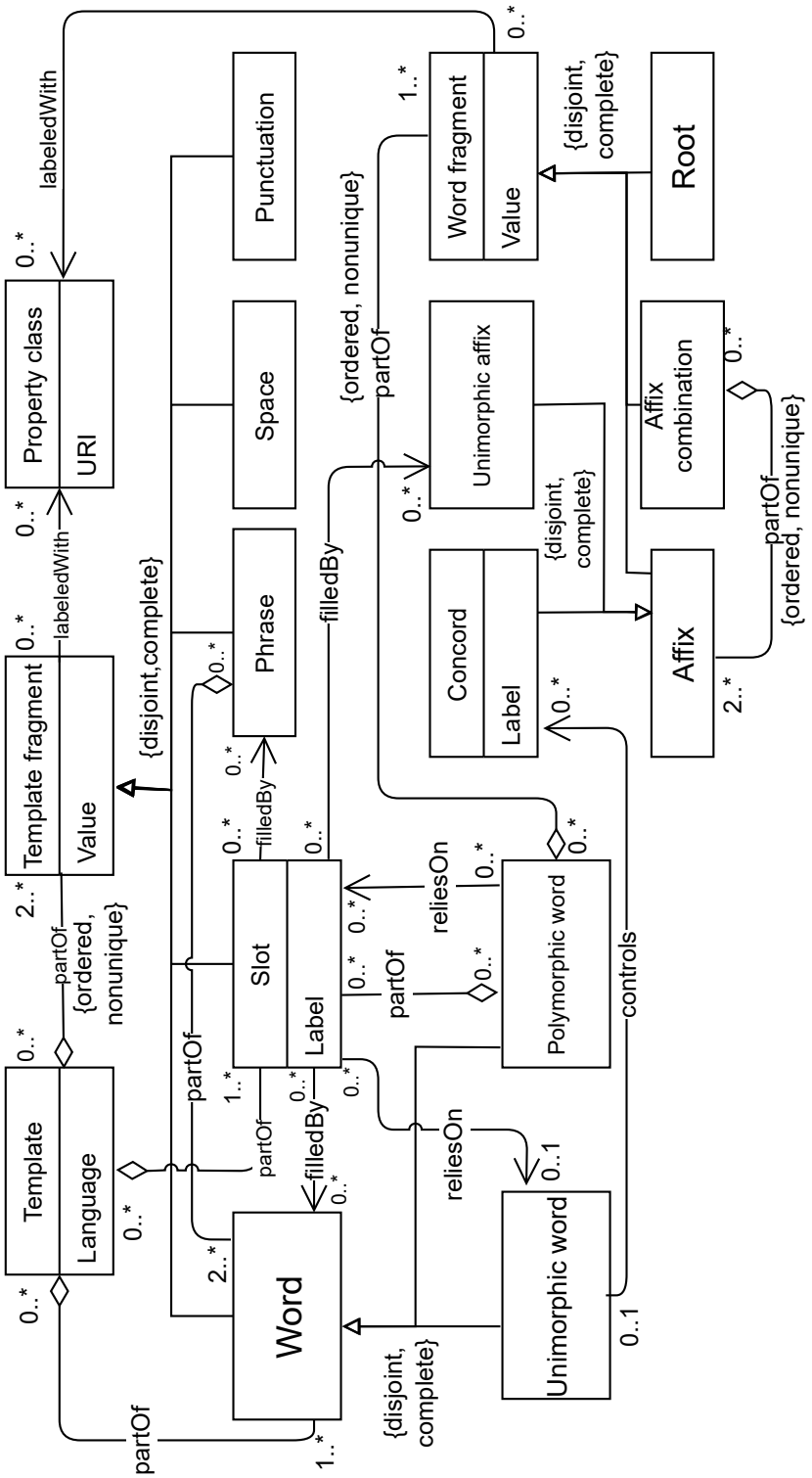


Figure 4.2: Representation of the concept, relations, and constraints found in the task ontology for template specifications using Unified Modelling Language (UML) notation.

- `Polymorphic word`: these are signs with multiple written representations because they possess at least one morpheme whose value is dependent on other words. For instance, the first word in the short French template "du/de la/des [l' animal]" is a polymorphic word that can have one of three written representations, depending on the number and gender of the noun inserted into the slot.

The two types of words can be used to form phrases, captured via the `Phrase` concept. Specifically, a `Phrase` is a sequence of `Unimorphic` and/or `Polymorphic Words`. Alongside these elements, we also have the notion of a slot via the `Slot` concept, to be used to be used to achieve dynamic strings. More precisely, a `Slot` is a placeholder in a template and its value is drawn from a finite collection of `Word`s or `Phrases` where the collection is referred to as slot fillers. The slot's label is a sign used to represent the role played by all the slot fillers. The label is not necessarily a word or phrase that exists in natural language. For instance, *[first_name]* is a valid slot label even though it does not exist in any natural language — only its constituent signs are valid English words (i.e., *first* and *name*).

Sub-lexical items There are three types of primary sub-lexical items in the model; `Root`, `Unimorphic affix`, and `Concord`. Their meanings are as follows:

- `Root`: this is a morpheme that carries the principal meaning of a word and it cannot be decomposed to finer granularity without losing its identity
- `Unimorphic affix`: this is a morpheme with a fixed value that carries some abstract meaning and it is distinct from the `Root`
- `Concord`: this is also a morpheme that is distinct from the `Root`, it carries some abstract meaning, it is appended to a root or stem for morphological agreement with other words, and its value is drawn from a finite collection of possible values (i.e., the concord fillers) based on the linguistic properties of a controlling `Unimorphic Word`. The concord's label represents the role played by all the concord fillers.

To demonstrate these concepts, we now turn to the pattern fragment that was presented in Chapter 1 and taken from by Keet and Khumalo's [136] work for verbalising universal quantification:

⟨ QC (all) for NC_1⟩onke ⟨N_1⟩

The elements that have been introduced can be used to capture this pattern. Specifically, in the above pattern, there is a `Word` element that may take different surface forms (e.g., *zonke, bonke, yonke,* etc 'all'). That element is followed by a `Slot` element that can be replaced by nouns. The `Word` element is made up of multiple sub-lexical elements. Specifically, it has a `Concord` and a Root element. The Root element has the surface *-onke* and cannot be decomposed further.

Similar to how the `Phrase` concept 'aggregates' the various Word concepts, the `Affix combination` concept can be used to 'aggregate' `Affix` concepts. Specifically, we define the Affix combination as being a sequence of `Affix`es. The properties of the various template and word portions can be specified via the `Property class`. The task ontology does not capture detailed subclasses of the `Property` class. Instead, we use two separate linguistic annotation ontologies for the concord types, one for isiZulu and another for isiXhosa, and they their knowledge was gathered from primary sources. This decision was made in order to avoid overburdening the task ontology, as depicted in Figure 4.2, with the linguistic annotations. The development and content of the two linguistic annotation ontologies will be discussed in Chapter 6.

Relations In the ontology, the `labeledWith` relation specifies the properties of a template/word portion. The `hasPart` is a type of mereological relationship that is equivalent to the inverse of Keet and Artale's [132] `part_of` relation. Since Keet and Artale have formalised the various part-whole relations in an OWL ontology that makes use of Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE), we could have reused it. We have chosen not to reuse their part-whole relations since the object property `part_of` is non-simple; hence, we cannot not use it in a minimum cardinality constraint in OWL 2. Cardinality constraints are important to detect template errors such as the invalid declaration of canned text as a template (e.g., a template must have at least one slot). The `filledBy` relation is a relation that specifies that a `Word` or `Phrase` is member of a specific slot's fillers and `controls` is a relation that specifies a rigid essential necessitation between a unimorphic word and a concord's value. The `reliesOn` relation is a relation that specifies a rigid existential necessitation between (1) a slot and unimorphic word or (2) between a polymorphic word and slot. In particular, in the first case it signifies that a polymorphic word inserted into the slot cannot have a value without a `Unimorphic word`. In the second case, the polymorphic word's value depends on the `Unimorphic word` that is inserted into the slot. For instance, if *[SC]onke* 'all' relies on the slot [subject], where SC stands for subject concord, then the first word can only have its final surface form after a

value is inserted into the slot. When the word *abantu* 'people' is inserted into the slot then first word's surface form will be *bonke* 'all'.

All these relations have inverses but they are not shown in the Figure 4.2 for brevity.

4.1.4 Formalisation

In order to take advantage of a reasoner for detecting and removing semantic inconsistencies, we formalise the model in OWL using Protege 5.5. We used FaCT++ 1.6.5 reasoner to detect possible inconsistencies and there were none found.

We demonstrate the translation from CQ to SPARQL query using the translation of Q1 to the following query:

```
SELECT distinct ?template (count(?unimorph) as ?numUnimorphWords)
(count(?phrase) as ?numFixedPhrases) ((?numUnimorphWords +
 ?numFixedPhrases) as ?totalFixedParts)
WHERE {
     {
     ?template a toct:Template .
     }
     UNION
     {
     ?template co:item ?unimorph .
     ?unimorph a toct:UnimorphicWord .
     }
     UNION
     {
     ?template co:item ?phrase .
     ?phrase a toct:Phrase .
     filter not exists { ?phrase toct:hasValue "" } .
     }
}
GROUP BY ?template
```

In the above listing, all unimorphic words and phrases are selected from the templates and the sums are calculated. Since most of the CQs, including CQ1 that we are using to demonstrate the CQ-to-SPARQL translation, pertain to information that can only be found in the ABox, we created a model to answer the CQs. The term 'Abox' refers to individual-level knowledge and goes hand-in-hand with the class-level knowledge (refereed to as the 'Tbox') in a knowledge base, as shown in Figure 4.3.

In our case, the model's TBox is made up of ToCT, linguistic annotation ontologies, and alignment between the two (`Property` $\sqsubseteq$ `PropertyClass`, `hasNounClass` $\sqsubseteq$ `labeledWith`, and `hasConcordType` $\sqsubseteq$ `labeledWith`). The purpose of the linguistic annotation annotation is to introduce language specific knowledge (e.g., isiXhosa

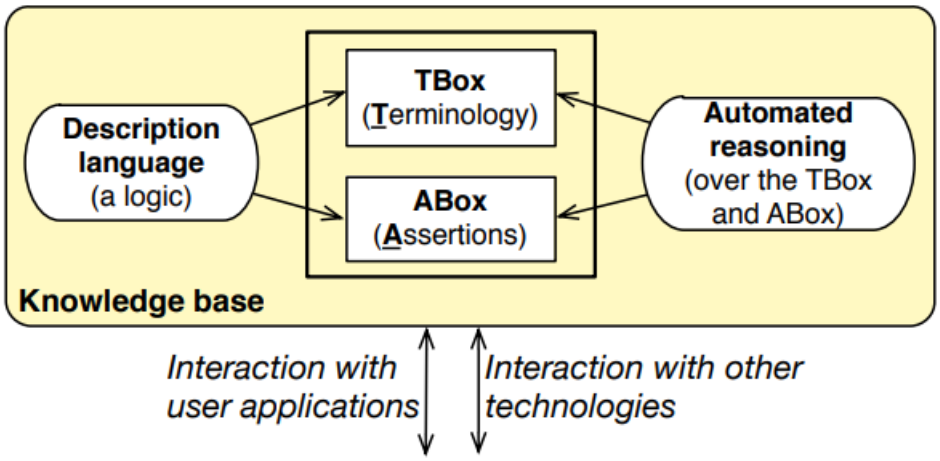


Figure 4.3: Conceptual representation of a Description Logic knowledge base (Source: [130, pg45]).

has 15 noun classes). We did not create a separate linking model, á la Ontologies of Linguistic Annotation (OLiA), since the created model's purpose is only to query establish the answerability of the CQs. The model's ABox is made up of the following template:

{personSlot} {SC} yeva nangoku

For the above ToCT template, we use an arrow from a concord to a slot to specify that its value depends on the noun inserted into that slot. We use curly braces to denote slot and polymorphic words are denoted via a box around its fragments. We also use the shorthand SC to refer to a slot that appears within a word and takes on values that are subject concords. The first slot is filled with unimorphic words that are nouns and those slot fillers are labelled in order to determine the noun class. The second word's concord is also labelled using the concord annotation ontology. Running the SPARQL query returns the result shown in Figure 4.4 where we can answer the competency question and determine that there is only a single segment that is fixed (i.e., 'nangoku').

Eight questions (CQs 1, 2, 3, 4, 5, 7, 11, and 12) were translatable and answerable after querying the model. Two questions (EQ8.1 and EQ8.2) were not translatable but were answerable through the analysis of the model: since templates and words are lists then the ordering is a property possessed by the members only in the context of the collection. Hence, the ordering information cannot persist if the collections do

template	numUnimorphWords	numFixedPhrases	totalFixedParts
hearingTemplate	"1"^^<http://www.w3.org/2001	"0"^^<http://www.w3.org/2001	"1"^^<http://www.w3.org/2001

Figure 4.4: List of fixed segments in the model used to answer the competency questions. This is the result of running the SPARQL query created from first competency question.

not exist. The remaining four questions (Q6, Q9, Q10 and Q13), were not answerable as they were no longer in scope. They pertain to linguistic rules that are have been deemed unnecessary for morphologically-enhanced templates. These questions were included as a means of filing the gap between the simple templates and the associable CGR. More specifically, they capture the use use of CGRs in a template and word and the order of CGR application. However, capturing this information may lead to complex templates and there is no benefit to capture this information via an ontology instead of the programming language used to build an NLG system. As such, we decided to no longer support their associated requirements.

4.2 Ontologies and models for concord annotation

When NLG systems that use ToCT are built, they will need to be able to resolve the values of concords, as they are found in various polymorphic words. To be able to resolve the correct value of a concord, they need to be able to determine the type of concord being used. ToCT's `labeledWith` can be used to specify a concord type, should a linguistic annotation ontology be chosen. However, the General Ontology for Linguistic Description (GOLD) and related artefacts have been demonstrated to not be up to the task [55]. Since the Noun Class System (NCS) ontology, and its corresponding isiXhosa and isiZulu axiomitisations, have already been created, we decided to build upon the resources and introduce related ontologies to be used for the linguistic annotation of concord types. Following the NCS ontology, we also used the framework presented in [55].

We gathered knowledge about the possible concord categories in Nguni languages from Katamba, Meeussen, Bourquin [128, 192, 34] to create stopgap linguistic annotation ontologies. The linguistic annotation ontology contains all the six shared concord types that are listed in Figure 4.5 and the *phi/mbi* concord types and subtypes of the *adjectival* and *relative* concords types. The isiZulu specific axiomitisation introduces constraints to specify that it does not have the *phi/mbi* concords or the subtypes of

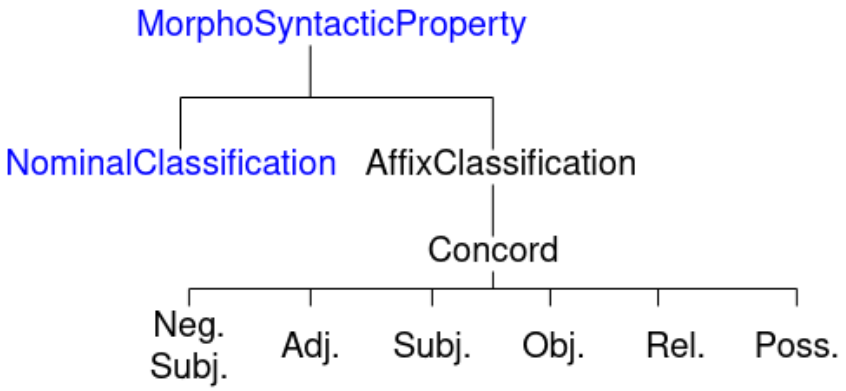


Figure 4.5: Informal list of the concords types found in the Concord Annotation Ontology. These are contextualised in relation to NCS's nominal classification property. Specifically, we used the color blue for NCS ontology's `NominalClassification` and `MorphoSyntacticProperty` concepts. (Abbreviations: Negative = Neg., Adjectival = Adj., Subjectival = Subj., Objectival = Obj., Relative = Rel., and Possessive = Poss.)

the *adjectival* and *relative* concords. A visualisation of the annotation ontology, in relation to the NCS ontology, is given in Figure 4.5.

The NCS and concord annotation ontologies can be linked/aligned ToCT via the following: `hasNounClass` $\sqsubseteq$ `labeledWith` and `hasConcordType` $\sqsubseteq$ `labeledWith`. We now turn to discuss how the various ontologies and models fit together and demonstrate the benefits of ToCT.

4.3 Use and benefits of artefacts

The relationship between the various artefacts is depicted in Figure 4.6. We use boxes to label the layers introduces in Chavula and Keet's framework [55]. The artefacts that are created by the present author are shaded in orange to differentiate them from existing ontologies and models. All the resources can be downloaded from `https://github.com/AdeebNqo/ToCT`.

We now use two templates taken from existing literature in order to demonstrate how to use ToCT. With respect to ToCT's applicability for Nguni languages and other languages outside that classification, we have chosen to capture isiZulu and Catalan templates. Our inclusion of Catalan is to demonstrate the usefulness of ToCT for moderately under-resourced languages that do not belong to the Nguni language group.

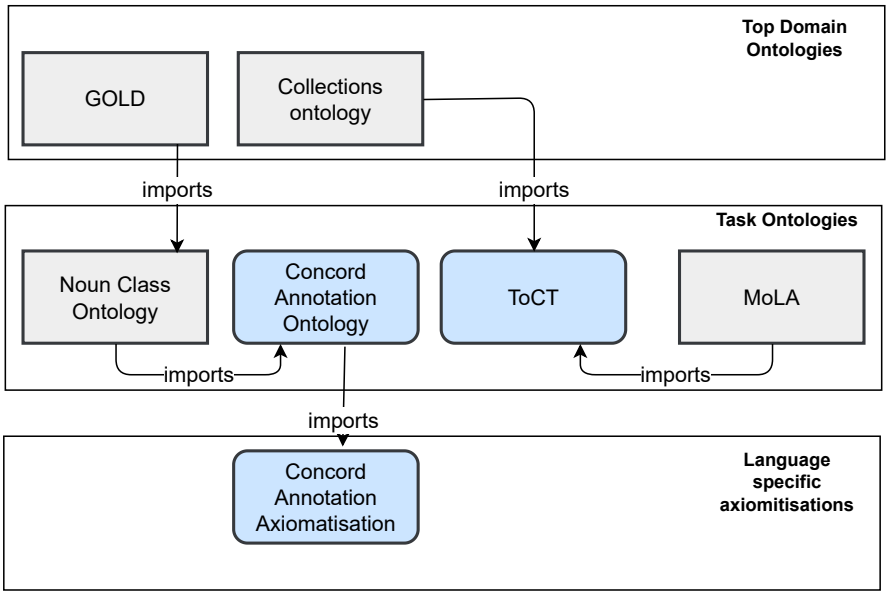


Figure 4.6: Representation of the various ontologies and models that are created and reused. Directed arrows labelled 'imports' denote that the source ontology or model is imported by destination artefact. We use blue boxes that have rounded corners for artefacts that we have created. We use grey boxes with sharp corners for artefacts that have been created by others.

4.3.1 IsiZulu

We encode a pattern taken from [136], which was created for verbalising OWL subsumption axioms, where the plain object properties are verbalised in the present tense. It generates texts of the sort *wonke umuntu udla isithelo esisodwa* 'each human eats at least one piece of fruit' where all the underlined sections are slots in the following template shown in Listing 6.2.3.1.

Listing 4.1: IsiZulu template for verbalising subsumption axioms (Source: [136])

```
⟨QC(all) for NC_x⟩onke ⟨ pl. N_1, is in NC_x⟩ ⟨ SC of NC_x⟩⟨verb stem⟩a
  ⟨N_2 of NC_y⟩ ⟨ RC for NC_y⟩⟨ QC for NC_y⟩dwa
```

We encode the above template given in Listing 6.2.3.1 using our task ontology and obtain the following partial template, expressed in Terse RDF Triple Language (Turtle) syntax (explained afterwards):

Listing 4.2: IsiZulu template captured using ToCT for verbalising subsumption axioms

```
<plainOPPresentTense> a toct:Template
     ; toct:supportsLanguage <kzn_zulu>
     ; toct:hasFirstPart <all>
     ; toct:hasLastPart <only>
     ; toct:hasPart <noun1Slot>, <opverb> , <noun2Slot> .
```

```
6
7  <kzn_zulu> a mola:Dialect
8       ; rdfs:label "isiZulu"@zu
9       ; mola:inFamily <isiZulu>
10      ; mola:inRegion <kzn> .
11
12 <all> a toct:PolymorphicWord
13      ; co:index '1'^^xsd:positiveInteger
14      ; toct:reliesOn <noun1Slot>
15      ; toct:hasFirstPart <quantConcord>
16      ; toct:hasLastPart <onke>
17      ; toct:hasNextPart <noun1Slot> .
18
19 <quantConcord> a toct:Concord
20      ; toct:hasLabel "quantConc"^^xsd:string
21      ; cao:hasConcordType <subjConType>
22      ; toct:hasNextPart <onke> .
23
24 <onke> a toct:Root
25      ; toct:hasValue "onke"^^xsd:string .
26
27 <noun1Slot> a toct:Slot
28      ; toct:hasLabel "noun1"^^xsd:string
29      ; toct:hasNextPart <opverb> .
30
31 <opverb> a toct:PolymorphicWord
32      ; toct:reliesOn <noun1Slot>
33      ; toct:hasFirstPart <subjConc>
34      ; toct:hasLastPart <aChar>
35      ; toct:hasPart <verbStem>
36      ; toct:hasNextPart <noun2Slot> .
37
38 <subjConc> a toct:Concord
39      ; toct:hasLabel "subjectConcord"^^xsd:string
40      ; toct:hasNextPart <verbStem> .
41
42 <verbStem> a toct:Slot
43      ; toct:hasLabel "verbStem"^^xsd:string
44      ; toct:hasNextPart <aChar> .
45
46 <aChar> a toct:UnimorphicAffix
47      ; toct:hasValue "a"^^xsd:string .
48
49 <noun2Slot> a toct:Slot
50      ; toct:hasLabel ""^^xsd:string
51      ; toct:hasNextPart <only> .
52
53 <only> a toct:PolymorphicWord
54      ; toct:hasFirstpart <relCOdwa>
55      ; toct:hasLastPart <odwa>
56      ; toct:hasPart <qualCOdwa>
57      ; toct:reliesOn <noun2Slot> .
58
59 <relCOdwa> a toct:Concord
```

```
60      ; toct:hasLabel "relativeConcord"^^xsd:string
61      ; cao:hasConcordType <relConType>
62
63  <possConType> a cao:RelativeConcord .
64
65  <qualCOdwa> a toct:Concord .
66      ; toct:hasLabel "qualitativeConcord"^^xsd:string
67      ; cao:hasConcordType <subjConType> .
68
69  <subjConType> a cao:SubjectivalConcord .
70
71  <odwa> a toct:Root
72      ; toct:hasValue "odwa"^^xsd:string .
```

The snippet given in Listing 4.2 is partial because we have left out the Space instances that should be placed between template fragments. The snippet uses three model prefixes, which refer to our task ontology (`toct`), the model for language annotations (`mola`), and the ontology we have used for isiZulu axiomitisation of the concord annotation ontology (`cao`). A concise visualisation of the souce of the various concepts found in the snippet is given in Figure 4.7.

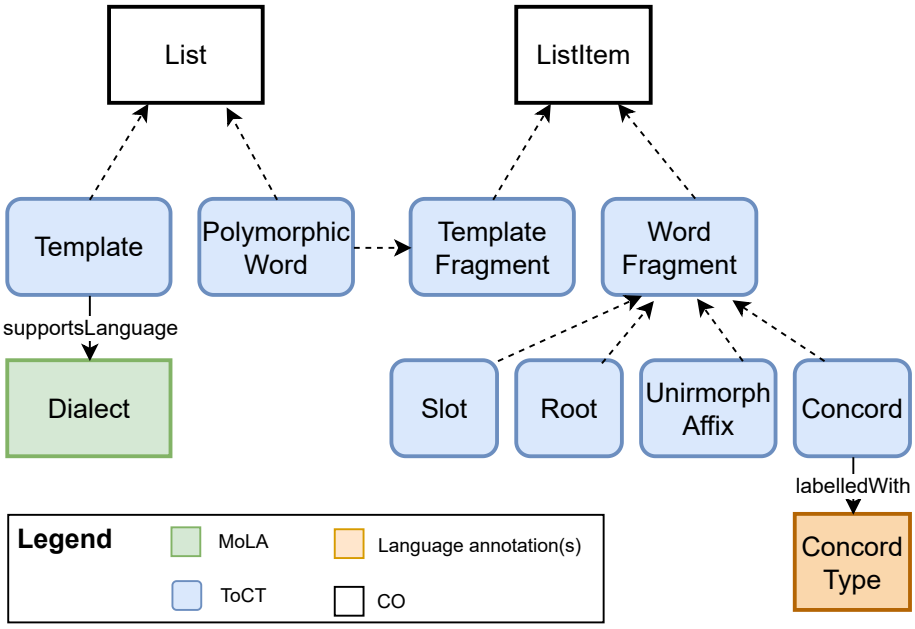


Figure 4.7: Representation of the classes used in the isiZulu and some of the relationships between them. We label the subsumption relations via dotted arrow where arrow's source is the subclass and its destination is the superclass. We also label the relationships that exist between the Template and MoLA's Dialect and Concord and the Linguistic annotation. All other relations are not shown for brevity. We use boxes with rounded edges to denote elements from ToCT and sharp corners for elements from MoLA (highlighted green), CO (highlighted white), and ontologies used for language-specific annotations (highlighted orange).

The isiZulu ToCT snippet illustrates the declaration of a template by showing three elements: (i) the main template container, (ii) partial details of the dialect supported by the template, (iii) and the template's words. With respect to the template's words, we will demonstrate how polymorphic words and slots are formed. More specifically, we will do this by emphasising how the various components of the two word types are created and how they differ to Keet and Khumalo's method of template capture.

The template's first and last words are specified through `hasFirstPart` and `hasLastPart` (lines 3-4), specialisations of Collections Ontology's `firstItem` and `lastItem`. In addition to specifying the first and last word, we also specify the ordering of words that fall in between them through the object property `hasNextPart` (see lines 17, 29, 36, and 51). The language supported by the template is specified via the `supportsLanguage` object property (line 2). The dialect supported by the template is captured in lines 7-10. It is an isiZulu dialect that is spoken in KwaZulu-Natal, South Africa (line 10). Other information that is currently encoded about the dialect is that it is part of the isiZulu family (line 9). A rich set of additional information can be provided via MoLA. This differs from [136]'s patterns in that the language, or dialect, that is supported by each template is made explicit.

The template's words are subclasses of `ListItem`, from the collections ontology, and they are specified as being contained in the template via `co:hasPart` (lines 5) thus the cardinality constraints can be enforced. Lines 12-17 illustrate the declaration of a `Polymorphic word`. The word is made up of two affixes, the so-called quantitative concord and the root '-onke' hence can result in the surface forms *zonke, bonke, lonke*, etc. The polymorphic word's value is determined by a value inserted into the slot `noun1Slot` since the polymorph is specified as relying on its form (line 14). Practically, the polymorphic word cannot have a final form unless a value that controls `quantConcord` is inserted into the `noun1Slot` slot. The index of the polymorphic word in the template is specified in line 13. That line, together with lines 5-6, need not be specified explicitly since they can be inferred by an automated reasoner when the first items are specified and the rest of the template fragments are connected via `hasNextPart`.

The quantitative concord associated with `all`, the first `PolymorphicWord`, is declared in lines 19-22. The concord has a label that can be used to visualise the template even when no values are inserted. For instance, if its labelled "QC(all)" the first word could be previewed as "[QC(all)]onke" when it is not tied to a specific noun. The concord's type is specified via `quantConType`, an instance of the quantitative concord

property from the isiZulu axiomitisation of the annotation ontology. The first and last words use `quantConcord` and `qualCOdwa` respectively and we have annotated those concords as being subjectival concords [34]. This differs slight from Keet and Khumalo's [136] use of a 'quantitative concord' since those words will be subjected to phonological conditioning rules to obtain the file word.

In the original isiZulu pattern, there are implied constraints that are necessary to generate the text. For instance, it is implied that in the slot that is present in the first word, one will use a morpheme whose value depends on other word. When patterns are used in practice, the authors tightly-couple the pattern and generation algorithm; hence, they can have an explicit enforcement of this constraint. Since we use a declaration specification for the simple templates, motivated by reuse and other features, we then need deliberate methods of detecting violations of template constraints. For instance, ToCT specifies that a template must have at least two fragments and a polymorphic word must have at least one fragment. As a such, if one would specify a template without elements, then an automated reasoner would detect the inconsistency. In our approach, the enforcement of template constraints does not come at expense of tightly-coupling the templates with text generation rules.

4.3.2 Catalan

We also encode a template taken from [118], which was created for verbalising uniqueness constraints in ORM diagrams. ORM diagrams are representations of the world that specify the objects and the relationships that exists between them — they are commonly used for capturing business rules. Its generates texts of the kind *Cada Persona ha de A com a màxim una Nom* 'Each Person must Has at most one Name' where the underlined part denotes a value that may take the value "a" or ∅, depending on the gender of the word's subject. The original template is given in Listing 4.3:

Listing 4.3: A Catalan template for verbalising uniqueness constraints in ORM

```
<Constraint xsi:type="Uniqueness">
<Text>-[Uniqueness] Cada</Text>
<Object index="0"/>
<Text>ha de</Text>
<Role index="0"/>
<Text>com a màxim un(a)</Text>
<Object index="1"/>
</Constraint>
```

We encode the template above using the task ontology and obtain the partial template, expressed in Turtle syntax, shown in Listing 4.4.

Listing 4.4: A Catalan template for verbalising uniqueness constraints in ORM captured using ToCT

```
1  <uniquenessORM> a toct:Template
2       ; toct:supportsLanguage <catalan>
3       ; toct:hasFirstPart <cadaPhr>
4       ; toct:hasLastPart <slot2>
5       ; toct:hasPart <slot1>, <haPhr>, <roleSlot>, <maxPhr>, <one> .
6
7  <cadaPhr> a toct:Phrase
8       ; toct:hasValue "-[Uniqueness] Cada"^^xsd:string
9       ; toct:hasNextPart <obj1Slot> .
10
11 <obj1Slot> a toct:Slot
12      ; toct:hasLabel "objectSlot"
13      ; toct:hasNextPart <haPhr>
14
15 <haPhr> a toct:Phrase
16      ; toct:hasValue "ha de"^^xsd:string
17      ; toct:hasNextPart <roleSlot> .
18
19 <roleSlot> a toct:Slot
20      ; toct:hasLabel "roleSlot"^^xsd:string
21      ; toct:hasNextPart <maxPhr> .
22
23 <maxPhr> a toct:Phrase
24      ; toct:hasValue "com a màxim"^^xsd:string
25      ; toct:hasNextPart <one> .
26
27 <one> a toct:PolymorphicWord
28      ; toct:reliesOn <obj1Slot>
29      ; toct:hasFirstPart <un>
30      ; toct:hasLastPart <suffix>
31      ; toct:hasNextPart <objSlot> .
32
33 <un> a toct:Root
34      ; toct:hasValue "un"^^xsd:string
35      ; toct:hasNextPart <suffix> .
36
37 <suffix> a toct:Concord
38      ; toct:hasLabel "[gender]"^^xsd:string .
39
40 <obj2Slot> a toct:Slot
41      ; toct:hasLabel "AnotherObjectSlot"^^xsd:string .
```

In Listing 4.4 we have left out the `Space` instances between template fragments in the snippet for brevity. In the snippet, there is a single polymorphic word (lines 8-12) that can have two forms. The word is composed of the `Unimorphic affix` whose label is “un” (line 10) and a `Concord` that can take the value ∅ or “a” (line 11)

depending on the value of the word put into the slot to which it depends (line 9). A Catalan specific ontology needs to be used to annotate the gender of words inserted into `obj1Slot` and a grammar engine can then determine the value of the `suffix` concord based on that gender. The correspondence between the two templates, by line numbers, are as follows:

1. Listing 4.3 Line 2 → Listing 4.4 snippet lines 7-9
2. Listing 4.3 Line 3 → Listing 4.4 snippet lines 11-3
3. Listing 4.3 Line 4 → Listing 4.4 snippet lines 15-17
4. Listing 4.3 Line 5 → Listing 4.4 snippet lines 19-21
5. Listing 4.3 Line 6 → Listing 4.4 snippet lines 23-38
6. Listing 4.3 Line 7 → Listing 4.4 snippet lines 40-41

There are three phrases in Listing 4.4 (`cadaPhr` in lines 7-9, `haPhr` in 15-17, and `maxPhr` in 23-25) and their text is captured via `hasValue`. One should not rely on only the `hasValue` property to specify its value. While the property is useful for provided an easy to find 'annotation' of the phrase's final value, one must also specify the phrases' associated word. We will demonstrate how `haPhr` can be specified, in a more detailed fashion, in order to take advantage of an automated reasoner. In order to prevent the creation of phrases with single or no words associated with them, then the `haPhr` element must be created as shown in Listing 4.5.

Listing 4.5: Detailed specification of a phrase for the Catalan template for verbalising uniqueness constraints in ORM captured using ToCT

```
1  <haPhr> a toct:Phrase
2       ; toct:hasFirstPart <hWord>
3       ; toct:hasLastPart <dWord> .
4
5  <hWord> a toct:UnimorphicWord
6       ; toct:hasValue "ha"^xsd:string
7       ; toct:hasNextPart <dWord> .
8
9  <dWord> a toct:UnimorphicWord
10      ; toct:hasValue "de"^^xsd:string .
```

Similar to the isiZulu example, ToCT is able to capture words whose form depends on other words. In the isiZulu ToCT template, we demonstrated that the word *-onke* 'all' is captured via a `PolymorphicWord` in lines 12-17 since it can take multiple forms (e.g., *zonke*, *bonke*, etc) depending on the noun class of the noun inserted into the template's first slot (i.e., `noun1slot` in lines 27-29). Similarly, the last word in line 6 of Listing 4.3 is captured via ToCT's `PolymorphicWord` class (see lines 27-31). A Catalan-specific ontology for linguistic annotation can be used to annotate some of

the words to be inserted into the `obj1Slot` slot. As a result, the NLG system that uses the Catalan ToCT template could produce the final value of the polymorphic word 'on the fly'. In addition, the values inserted into slots can also be captured via the `PolymorphicWord` concept. As such, the ToCT template would not generate the incorrect Catalan text *Cada Persona ha de A com a màxim una Nom* 'Each Person must Has at most one Name' where the inclusion of the underlined part is dependent on the word being quantified.

4.3.3 Other languages and benefits

In the isiZulu example, we have presented an automated reasoner as a beneficial tool for detecting violations of template constraints. That ability is crucial for morphologically rich languages since there might be inconsistencies introduced when creating multi-part concepts, such as polymorphic words. In addition to that, automated reasoners can be used with ToCT to support the ability to share and reuse templates for any language, especially in real-world systems where one is likely to have a large repository of templates. In such cases, one needs to have the ability to search for potential templates to reuse by using the 'meaning' of the template fragments. To demonstrate the benefits of the aforementioned ability, consider the following scenario: we are interested in building a UML verbaliser and has access to Jarrar et al.'s ORM verbalisation templates and several other templates that we created. All the templates in the collection are specified using ToCT. The person would like to answer the following question: *Which templates, in the collection, can I reuse as-is or in a modified form to verbalise UML diagrams?*

Since the ToCT allows the annotation of templates using the `labeledWith` relation, one can specify the 'meaning' of certain fragments (e.g., slots). For instance, using the snippet in Listing 4.4 that was introduced in Section 4.3.2, the instances `obj1Slot`, `roleSlot`, and `obj2Slot` can be labelled, using some annotation ontology, to identify that they expect ORM objects and roles. Using a meta-model for conceptual models, the information system used to manage the template collection can determine the correspondence between ORM and UML. Thus suggest templates that may be appropriate for reuse, even if they were initially designed for a different type of concept model. This feature is not limited to the discovery of templates for verbalising conceptual models. Technically, one can achieve the same goal by annotating XML templates, however, ToCT goes beyond that and caters to languages that are morphologically rich and can rely on the existing reasoners.

4.4 Discussion

Having introduced, evaluated, and demonstrated how to use ToCT, we now turn to discuss the strengths of our approach compared to existing methods. There are no existing ontologies for template specification other than ToCT. English relies on straightforward templates, since it has relatively simple grammar and disjunctive orthography, and thus can thrive well without an ontology-driven solution.

In this chapter, we presented ToCT to support ontology-driven solutions. While the focus was on catering for templates for African languages, the plurality of incompatible verbalisers for English suggests it can benefit from at least systematization, if not standardisation. ToCT can assist with that and its strengths lie with grammatically more challenging languages with features such as agglutination and having to deal with gender and noun classes that affect other words in a sentence. More specifically, it does not require the creation of multiple forms of the same template to capture changing words. It also does not pose a categorical limit for slot fillers; any affix, word, or phrase can be used and their POS or sub-lexical category can be specified separately via the `Property class` and `labeledWith` object properties (cf. pattern's [133] original coverage of only nouns, verbs, and pronouns, but not, e.g., attributes). Consequently, with ToCT, subject domain semantics are not buried in the code and there is an explicit template specification thus enabling reuse.

In accordance with Findable, Accessible, Interoperable, Reusable (FAIR) principles, the ontology and supplementary material such as SPARQL queries associated with the CQs have been assigned a license (CC BY 4.0), have a persistent identifier[3] and metadata, and are findable. Interoperability is fostered by formalising the template model in a Semantic Web language. We recognise that ToCT captured templates are long and some of its concepts, especially ones with multiple parts, are not easy to read and create by hand. For instance, the ToCT-captured Catalan template is longer than Jarrar, Keet, and Dongilli's [118] XML template. While this is often the case for ToCT templates, it is trumped by ToCT's ability to support grammatically complex languages, automated reasoning (cf. existing models, such as Jarrar, Keet, and Dongilli's XSD schema), and interoperability, among other benefits. Moreover, the effort required for creating, managing, and reading ToCT can be reduced in the long term since Graphical user interface (GUI)-based tools can be created for template induction, management, discovery, and visualisation. Such interfaces can

[3] `https://doi.org/10.5281/zenodo.4704362`

be thought of as being analogous to block-based visual programming language tools (e.g., Google's blockly[4] library and editor). This means that NLG engineers do not need to be highly skilled to create templates. They would only need to understand the meaning of ToCT's concepts. Theoretically, it should be easier to learn and create ToCT templates when using such tools, compared to creating large grammars. This is similar to how it can be easier to work with domain-specific languages vs. general purpose programming languages since the former are specialised.

In the isiZulu and Catalan examples presented in Section 4.3 to demonstrate how to use the ontology, the rules for deciding the value of each concord, based on the word that governs it, and the rules that determine the formation of each polymorphic word are concord value resolution are not encoded in the templates. The CGRs are purposefully kept separate so that they are usable for different applications' templates. This raises the question of how to organise the various components in order to ensure maintainability and resource reuse. Economical use of resources and general maintainability must be key considerations because Nguni languages are low-resourced. Since our goal is create the foundations for how build surface realisers that are reusable and easy to maintain, we now turn to architectural considerations in the next chapter.

[4] `https://developers.google.com/blockly`

Chapter 5

Knowledge-driven architecture for a maintainable surface realiser

In the previous chapter, we presented a task ontology for capturing morphologically rich simple templates. The templates, captured via the task ontology, can be used together with other components. This raises the question of how best to organise the various components in a surface realiser. The few attempts at building realisation components for African languages [129, 44] have been *ad hoc* and have not made architectural design decisions a centrality; hence, their maintainability is negatively affected.

A maintainable surface realiser for Niger-Congo B languages could be created by using an existing consensus architecture (i.e., one where there is wide agreement in the NLG field regarding what modules ought to exist and how they should be organised.), as-is or in modified form. However, there is no consensus in Natural Language Generation (NLG) on what tasks should be in the surface realisation module and how they should be organised. The closest thing to a consensus is Reiter and Dale's three-step pipeline architecture [241, 68] and the Reference Architecture for Generation Systems (RAGS)[1] project showed that, in practice, researchers who choose the three-step architecture do not agree on where to place tasks between the modules [196]. For instance, while Reiter and Dale present the surface realisation module as only being responsible for ordering, several NLG systems also make lexicalisation the module's responsibility. Overall, this means that the following statements hold:

- Existing African language NLG systems do not use an architecture that yields realisers that are easy to maintain.

[1] `http://mcs.open.ac.uk/rags/`

- There is no consensus on a maintainable surface realiser architecture within NLG that can be tweaked for Niger-Congo B (NCB) languages.

In this chapter, we address this gap for NCB languages by conducting an extensive literature review and comparison of 77 reviewed surface realisation systems. Instead of using the low-level tasks listed in [68, 196], we focus on five aspects: tactical decisions, structure encoding, structure induction, structure linearisation, and candidate ranking. Our tasks differ from [68, 196] in that they are more detailed and therefore enable a fine-grained analysis of surface realisation. We focus on these tasks based on our experience with the control issues and tight coupling of linearisation rules by systems that we evaluated. Examples of the evaluated systems are the isiZulu verbaliser [138] and Komet-Penman Multilingual (KPML) [18]. These issues affect the ease of use and reuse of the few existing resources. Using this comparison of existing surface realiser modules, we identify their limitations for NCB languages and then proceed to the creation of a new architecture for surface realisers to support low-resourced NCB languages. The architecture possesses a majority of the maintainability characteristics specified in the software product quality model presented in [38] (namely modularity, componential re-usability, and analysability). Our architecture differs from existing ones because of the following:

- It introduces a pre-processing module that operates on sentential structures in order to introduce context-specific grammatical information.
- It places the ranking module prior to linearisation so that it can have access to explicit grammatical knowledge/annotations.
- It uses an ontology to formalise templates and therewith allows analysability.
- It offers detection of logical inconsistencies in templates, sharing, and comparison of templates.
- Lastly, it moves tactical decisions outside the realiser.

Section 5.1 presents the method used to collect systems for analysis, Section 5.2 introduces the criteria used for the analysis of the architectures, Section 5.3 presents our categorisation of the architectures used in existing systems, Section 5.5 presents the new architecture, Section 5.6 compares the new architecture with existing architectures, Section 5.7 discusses the utility of using an ontology to capture template knowledge, and Section 5.8 concludes and summarises the chapter.

The work presented in this chapter was published in the *ACM Transactions on Asian and Low-Resource Language Information Processing* [170]. We have updated it and used a different set of images to improve clarity.

5.1 Collecting surface realisers for analysis

We review existing surface realisers as they are found in multiple domains and eras in order to identify current and past architectural trends at finer detail. This is done to uncover knowledge that is useful for building maintainable architectures for NCB languages. We compiled a list of 77 NLG systems and tools in preparation for the architecture analysis, by extending the 54 verbalisers and tactical realisers found in Chapter 3 with the following resources:

- 9 grammar engines (SimpleNLG variations, GenDR [151], and JSrealB [200]).
- 12 data-driven NLG systems (10 of which are recent and 2 are dated).
- 2 recent systems that make use of an augmented template tool (e.g., RosaeNLG).

We do not include the submissions to the surface realisation tasks since they are not developed with maintainability in mind. The submissions were developed with an emphasis on relying on the same kind of input and making it possible to use automated metrics for comparison. This was done because the goal of the task "was to make it possible, for the first time, to directly compare different, independently developed surface realisers by developing a 'common-ground' input representation that could be used by all participating systems to generate realisations from" [23], even though "[b]y the time teams submitted their system outputs, it had become clear that the inputs required by some types of surface realisers were more easily derived from the common-ground representation than the inputs required by other types. There were other respects in which the representations were not ideal, e.g. the deep representations retained too many syntactic elements as stopgaps where no deeper information had been available" [22].

From the above initial list, we manually read all the papers and then removed systems whose paper(s) have insufficient details about the surface realiser's architecture to be able to determine how it works (e.g., because the paper's scope was different), and then manually determined the architectures used by the remaining systems.

We constructed the above list so as to have surface realisers that are used in-the-wild and analyse their architectures. Overall, this review of systems differs from the one presented in Chapter 3 because of the following:

1. It does not focus on existential relationships between the templates and grammar rules
2. It analyses all kinds of surface realisers that are used by real-world systems and it is not limited to the analysis of grammar-infused templates and systems that make use of such templates.

Instead, this review analyses a number of surface realisation tasks and their organisations in various systems. The criteria used in the present work is introduced in the next section.

5.2 Analysis criteria

The current work's comparison focused on the following aspects:

- Structure selection: This is the tactical decisions component, which acts as the interface between the surface realiser and prior modules (i.e., it connects the various modules). It is responsible for making linguistic decisions given the semantic input (e.g., deciding on syntactical structure to use for a certain event type) [94]. We use the term 'structure selection' instead of tactical decisions in order to improve clarity regarding the relationship of this task with structure encoding, induction, and linearisation. Moreover, this task can be thought of as a kind of "structure determination". However, we prefer to use the term "selection" to make clear that it may involve the selection from an existing set of structures. Using SimpleNLG [94] as an example, the term "determination" may suggest that the NLG engineer is responsible for the creating English grammar rules for encoding sentential structure and using them at the same time. However, SimpleNLG already provides such rules and the engineer is only selecting which ones are appropriate for certain semantic input to their system.
- Structure encoding: This is the method used to encode the sentential structure. It captures the elements and position(s) of elements that will be inserted in order to form the final surface text, and specifies how they elements are to be ordered. For instance, some realisers use a template while others may use a phrase structure tree.

- Structure induction: This refers to the method used to create the structures used for capturing sentences. These structures come in many forms. For instance, they could be phrase-structure trees or templates, among the many possible options. In some cases, the structures may be induced from examples, but they can also be induced/created manually by the NLG engineer based on their domain expertise.
- Structure linearisation: This is the formation of text from some ordering structure. In the case of simple templates, it is equivalent to slot filling. In the case of grammar-infused templates and grammar-only approaches, it may also include forming words from lemmas, and traversing a tree to form the final surface texts.
- Ranking: The candidate filtering mechanism is responsible for selecting one sentence/structure out of many candidate output sentences or sentential structures.

We have chosen the above tasks for analysis so as to zoom into overlooked issues and avoid limitations seen in architectures of previous-generation systems. For instance, structure selection is important because analysis of the connection between the preceding module and the realiser provides us with an understanding of how to avoid the control issues present in dated wide-coverage tactical generators [94]. More precisely, such systems require a specialised input form since they make tactical decisions the responsibility of the realiser. As such, they offer no direct way for control over how "phrases are built and combined, inflectional morphological operations and linearisation" [94, p91]. In addition, they allow us to scrutinise the placing of the linearisation algorithm, and its combination with other aspects, is key in avoiding the need to create an algorithm each time a new template is created (e.g., [129, 44]). Our analysis of the method used to encode the sentential structures is important because some techniques (e.g., use of large-scale grammars) are not suitable for low-resourced languages. Lastly, we include the ranking because it is intertwined with linearisation in some NLG systems.

5.3 Surface realiser architectures and categories

The complete list of considered systems and tools are given in Table 5.1. The twelve architectures differ in the organisation of the various modules and methods employed for structure induction, encoding, and linearisation.

[2] `https://github.com/m477301/KnowledgeGraphVerbalizer`

Table 5.1: List of all classified systems whose surface realiser where classifiers and their corresponding architecture identifiers. Abbreviations: Architecture = Arch. and Identifier = id.

Arch. id.	**Reference(s) to system**
0	Aguado et al. [3], Bateman [18], Bouayad-Agha et al., Bouayad-Agha et al. [33, 32], Coch [61],, Dongilli and Franconi [79], Lareau et al. [151]
1	Cimiano et al. [60], Lavoie and Rainbow [153] Lavoie and Rainbow [153]
2	Bohnet et al. [27]
3	Busemann [40]
4	Knight and Hatzivassiloglou [140], Langkilde [148]
5	Nakanishi, Miyao, and Tsujii [205], White [283]
6	Angeli, Liang, and Klein [10], Kondadadi, Howald, and Schilder [144]
7	Byamugisha, Keet, and DeRenzi [44], Keet, Xakaza, and Khumalo [138], Lyudovyk and Weng [163]
8	Androutsopoulos, Lampouras, and Galanis [8], Ang, Kanagasabai, and Baker [9], Camilleri, Fuchs, and Kaljurand [47], Casteleiro et al. [50], Dannélls et al., Dannélls [71, 70], Davis et al. [72], Davis et al. [73], Elhadad and Robin [86], Gruzitis, Nespore, and Saulite, Gruzitis, Nespore, and Saulite [100, 101], Hossain, Rajan, and Schwitter [112], Jarrar, Keet, and Dongilli [118], Kaljurand and Fuchs [124], Lim and Halpin [160], Liang, Stevens, and Rector [157], Liang et al. [159], Hewlett et al. [108], McRoy, Channarukul, and Ali [190], Sanby, Todd, and Keet [249], Stenzhorn [263], Stevens et al. [264], Weal et al. [280]
9	Amith et al. [7], Bollmann [28], de Oliveira and Sripada [74], Dušek [83], Gatt and Reiter [94], Hielkema, Mellish, and Edwards [109], Kuanzhuo, Lin, and Zhao [146], Mazzei, Battaglino, and Bosco [186], Molins and Lapalme [200], Ramos-Soto, Gallardo, and Diz [235], Vaudry and Lapalme [277]
10	Lavoie and Rainbow [153]
11	Mairesse et al. [175], Moryossef, Goldberg, and Dagan [202], Wong [289]
12	Castro Ferreira et al. [51], Gubbala et al. [102], KnowledgeGraph Person verbaliser[2]

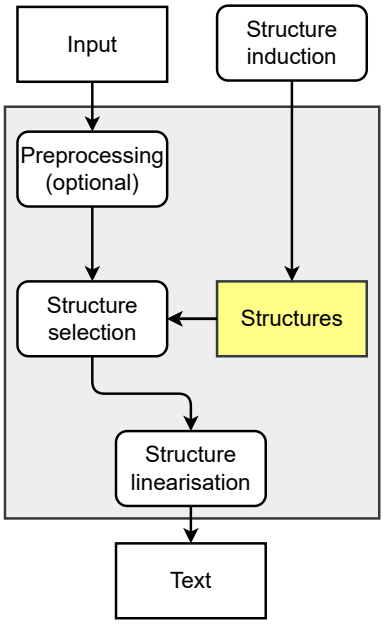


Figure 5.1: Representation of surface realiser architecture category 1. We use the grey box with sharp corners to denote the surface realiser and a yellow box with sharp corners to denote that the structures are captured using a grammar.

We first group these architectures into categories based on similarities in their organisation of modules and this resulted in six categories — they are given in Table 5.2. While our use of the term 'architecture' is in the usual sense, we recognise that use of the term 'architecture category' is not common place; hence, we now turn to demonstrate how to interpret Table 5.2 using the system described by Lavoie and Rainbow [153].

The system uses architecture 10 and according to Table 5.2, it belongs to category A4. Its surface realisers only contain sentential structures and a structure linearisation module. Table 5.2 also shows us that unlike other architectures in the same category, it relies on hand-coded sentential structures and uses grammar rules for linearisation.

We now turn to discuss these categories and provide examples of systems whose realiser's belong to each of the categories. Our examples do not put a lot of emphasis on categories A1-2 since those architectures are only prevalent in dated NLG systems.

Architecture category 1 (AC1): tactical generators The first category of surface realiser architectures, referred to as the AC1 category, is shown in Figure 5.1. It

Table 5.2: Surface realiser architectures, their associated categories, and the differences between them. Abbreviations: Select = Sel., Ranking = Rank., and Structure = str.

Architecture identifier	**Category**	**Structure induction**	**Structure linearisation**	**Module A**	**Module B**
0	AC1	Hand-coded	Grammar	-	-
1	AC1	Data-driven	Rule-based	-	-
2	AC1	Data-driven	Data-driven	-	-
3	AC1	Hand-coded	Rule-based	-	-
4	AC2	Hybrid	Data-driven	Str. sel.	Str. lin. + rank.
5	AC2	Hand-coded	Rule-based	Str. sel. + lin.	rank.
6	AC3	Data-driven	Rule-based	Str. sel. + rank.	Str. lin.
7	AC3	Hand-coded	Rule-based	Str. sel. + lin.	$\emptyset$
8	AC3	Hand-coded	Rule-based	Str. sel.	Str. lin.
9	AC4	Hand-coded	Rule-based	-	-
10	AC4	Hand-coded	Grammar	-	-
11	AC5	-	-	-	-
12	AC6	Hand-coded	Rule-based	-	-

is used by so-called tactical generators; realisers are responsible for making linguistic decisions given semantic representations and also applying linguistic rules to convert decided upon syntactical structures to obtain surface text (see [94] for details). Tactical decisions are conducted via rules. Sentential structure encoding and linearisation is conducted via a large grammar and the formalism used for the grammar varies across the various systems. An example of such a tactical generator is the KPML system [18].

Architecture category (AC2): tactical generator with ranker The second category, referred to as AC2, is shown in Figure 5.2. It is similar to AC1 in that it makes use of a grammar for encoding sentential structure. However, it differs because it has a sentence ranking module. The architectures that belong to AC2 differ since some use a data-driven module and other use rules for structure linearisation (see Table 5.2). For instance, in one system [271] the authors performed a comprehensive analysis of a parallel corpus made up of pollen concentration data and human-authored pollen reports. They analysed those reports in order to create an NLG system that abides by the pipeline architecture introduced in Dale and Reiter [68]. For surface realisation, their system reuses the approach taken for the Sumtime project, in that they make use rules to order "the phrases in the output and also to perform punctuation tasks" [259]. A different approach can be seen in Knight and Hatzivassiloglou's [140] work where they take the input, construct a lattice (i.e., a finite state machine), and use a bigram model to determine the most likely path in the lattice. Specifically, instead of creating grammar rules or templates to verbalise each input, they use the input to construct the a lattice that specifies all the possible output sentences (some of which are not grammatically correct) that can be generated for an input. Once a lattice exist, they then make use of a bi-gram model to rank the possible paths.

There are modules labelled A and B in category AC2. They represent the different ways that the two categories combine structure, linearisation, and ranking. The architectures that belong to this category are illustrated in Figure 5.3. In the category, structure linearisation is either combined with selection in module A or with ranking in module B. Technically, we could have split architecture 4 and 5, as labeled in Figure 5.3, such that they are separate architecture categories; however, doing so adds no explanatory value. Another example of a generator that uses the architecture is the generator built by [148], that takes semantic input and uses it to create a forest via rules (a generation forest is a context free representations that specifies all the

possible sentences that can be generated). The forest's subtrees are ranked and then linearised to produce the final sentence.

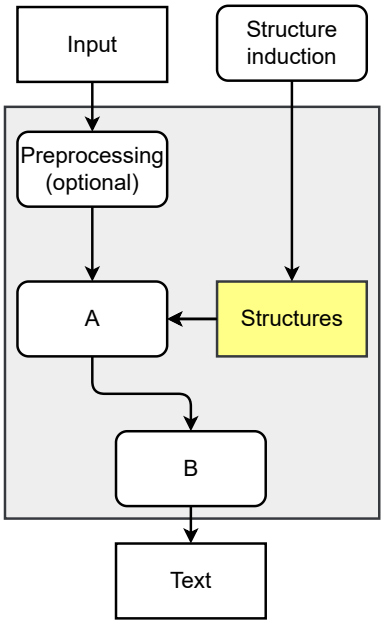


Figure 5.2: Surface realiser architecture category AC2. The modules labelled A and B represent different combinations of structure selection, linearisation, and ranking. For instance, in architecture 4 we have A = structure selection and B = structure linearisation and ranking. We use the grey box with sharp corners to denote the surface realiser and a yellow box with sharp corners to denote that the structures are captured using a grammar.

Architecture category 3 (AC3): structures in imperative languages The AC3 category, shown in Figure 6.1, is similar to AC2, since some of its architectures include a ranking module. It also has modules labelled A and B and they also represent the different ways that the two categories combine structure, linearisation, and ranking. In category AC3, there is no ranking and linearisation either can be combined with selection in one module (A) or done separately (in A and B, respectively). Similar to category AC2, we could have split these into separate architecture categories, however, doing so would add no explanatory value. The category also differs from AC2 as it relies on templates for structure encoding. A visualisation of the architectures that belong to this category is given in Figure 5.5.

An example of such an NLG system is the template-based OWL verbaliser for Afrikaans [249], where the tactical decisions component maps the various axiom types that are supported to templates. For instance, axioms of the type SubClassOf(C1 C2) (in OWL

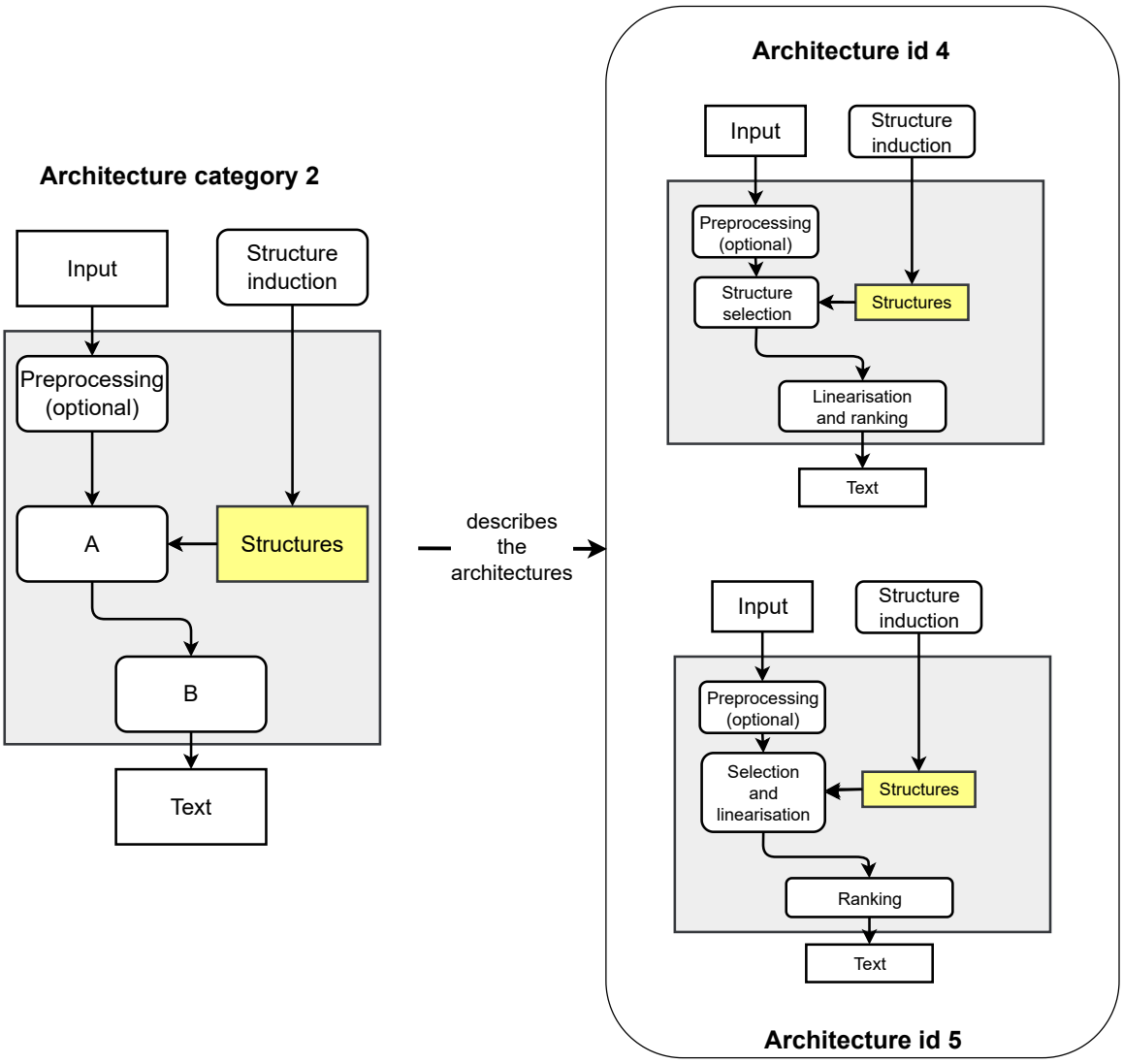


Figure 5.3: The two architectures that belong to architecture category 2

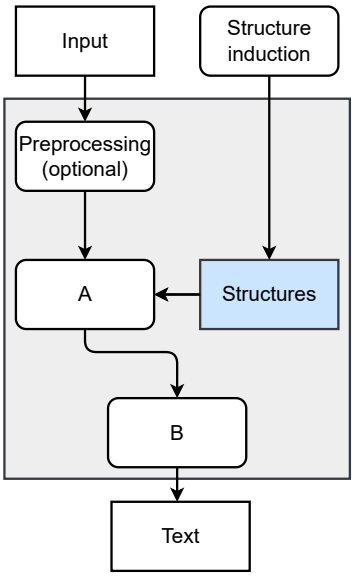


Figure 5.4: Surface realiser architecture category AC3. The modules labelled A and B represent different combinations of structure selection, linearisation, and ranking. For instance, in architecture 6 we have A = structure selection + ranking and B = structure linearisation. We use the grey box with sharp corners to denote the surface realiser and a blue box with sharp corners to denote that the structures are captured using a template that may be grammar-infused.

functional syntax style) where C1 and C2 are classes, are mapped to a single template. This mapping is encoded via a Java rule that selects the following template whenever it encounters a subsumption axiom[3]:

```
<Constraint type="OWLSubClassOfAxiom">
<Text>Elke</Text>
<Object index="0"/>
<Text>is 'n</Text>
<Object index="1"/>
</Constraint>
```

When given SubClassOf(DOG, ANIMAL), the verbaliser will produce *Elke hond is 'n dier* ('Each dog is an animal'). The linearisation of each template is done within Java; it involves filling in slot values and doing minor cleanup such as removing trailing spaces. The system described by [144] also uses this architecture, but it uses a data-driven module for ranking and selecting the best template to use for any provided input. The chosen template and processed input are then linearised using a rule-based module.

[3]Taken from `https://projects.cs.uct.ac.za/honsproj/cgi-bin/view/2015/sanby_todd.zip/index.html`

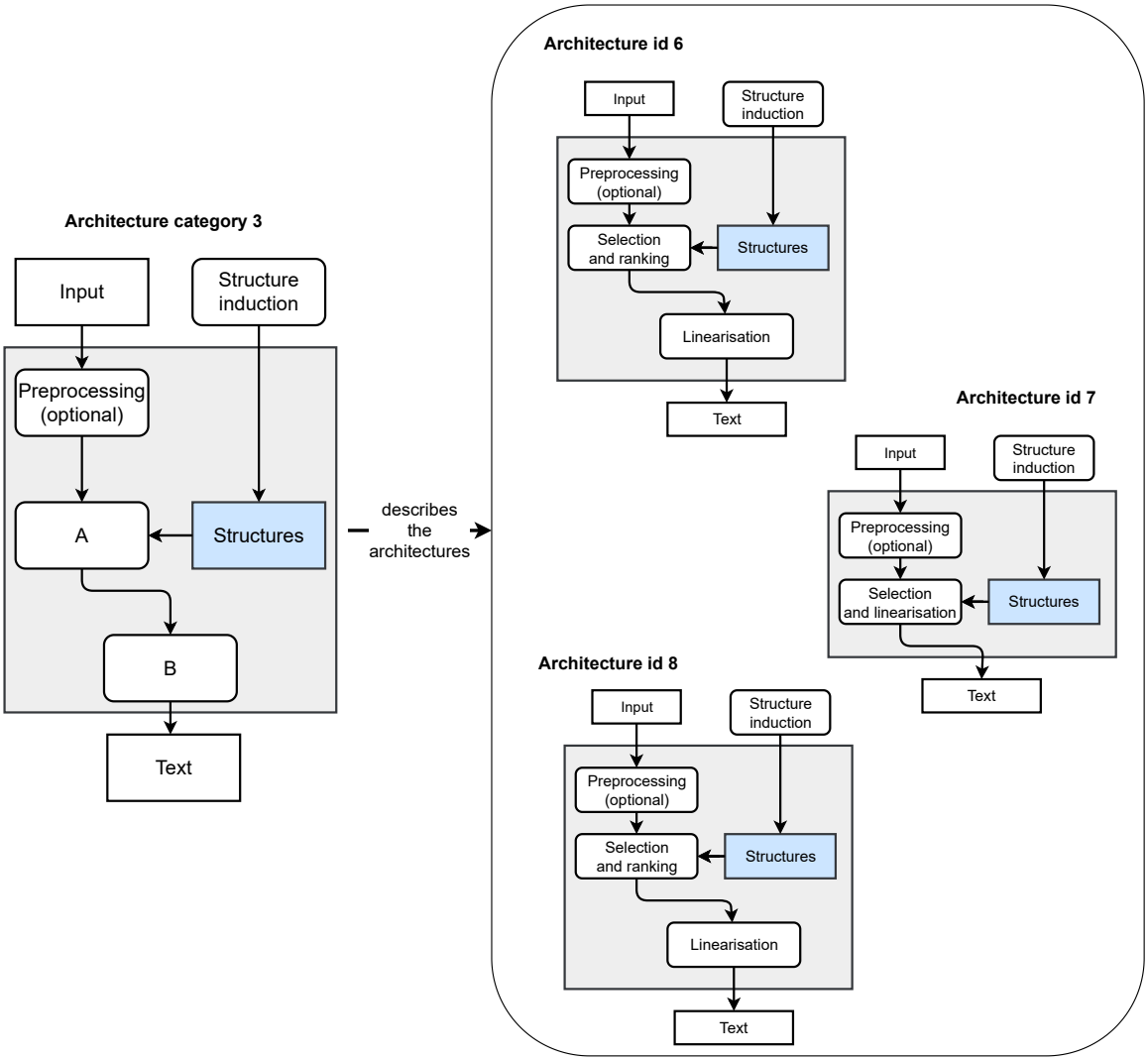


Figure 5.5: The three architectures that belong to architecture category 3

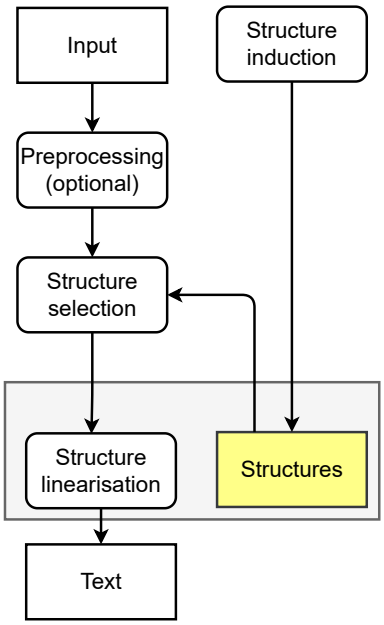


Figure 5.6: Representation of surface realiser architecture category AC4. We use the grey box with sharp corners to denote the surface realiser and a yellow box with sharp corners to denote that the structures are captured using a grammar.

Architecture category 4 (AC4): grammar engines The AC4 category, shown in Figure 5.6, is used by most contemporary NLG systems. These systems rely on grammar engines such as SimpleNLG [94] (or its adaptations for other languages such as Tibetan [146]). The engine provides reusable language-specific rules for creating syntactic structures and linearising them so as to obtain well-formed text. The choice of which syntactic structure to use given the semantic input is not made by the realiser. Instead, it is left to the discretion of the engineer, i.e., the realiser makes no tactical decisions. As such, the input is a sentence plan (i.e., an underspecified syntactic structure) or the intermediate representation and a choice of syntactic structure that is to be created by the realisation engine. Based on the input, control is given to the engine to create the final syntactic structure and apply all necessary linguistic rules. An example of a system that follows this architecture is [185], who feed Content MathML (CMML) expressions to the sentence planner and it detects which expression categories are found in the input. For instance, when provided with the CMML expression for $\{x \mid x \leq 0\}$, the planner detects that it is made up of the *relational*, *(arithmetic, algebraic, set)*, and *conditional set*. Using the predefined rules for mapping each category to a syntactic structure, it creates a structure for the complex category (i.e., the *conditional set*) where the internal nodes capture the simple categories (i..e, the *relational* and *(arithmetic, algebraic, set)*), the child

nodes capture lemmas, and edges capture dependency relations between the various nodes. This sentence plan is passed to the system's coordinator, which then builds a constituency-based syntactical structure using SimpleNLG-it [186] that corresponds to the input and then linearises the tree into text.

Architecture category 5 (AC5): data-driven and integrated systems The AC5 category, shown in Figure 5.7 does not have distinct module for making tactical decisions to determine a natural language's syntactical constraints, it has no explicit sentential structure encoding and linearisation module. Instead, it makes use of a data-driven model (most contemporary systems use a deep neural network) to map some intermediate representation to natural language text. The first group of systems that belong to this category do not have an identifiable surface realisation component, even though they are modular. This is because their modules do not focus on the criteria introduced in Section 5.2. For instance, Moryossef et al. [202] trains a sequence-to-sequence model with attention for converting text plans to surface text using the OpenNMT toolkit. A text plan is a list of sentence plans where each sentence plan is a labelled directed graph where the nodes represent concepts and direction edges represent predicates. So when the trained neural network is given the sentence plan [(John, residence, London), (England, capital, London)], it may generate "John lives in London, the capital of England" [202]. The authors generated the texts using the probability distribution $p(s|t) = \prod_{i=1}^{n} p(t_i|t_1, \ldots, t_{i-1}, \mathbf{s})$ where $\mathbf{s} = s_1^n$ and $\mathbf{t} = t_1^m$ are the plans and output texts respectively. The value of $p(t_i|t_1, \ldots, t_{i-1}, \mathbf{s})$ is is approximated using the model described by Bahdanau, Cho, and Bengio [13], with some minor modifications (see [202]).

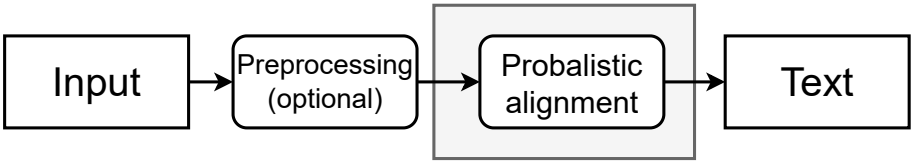


Figure 5.7: Representation of surface realiser architecture category A5. We use the grey box with sharp corners to denote the surface realiser.

The only neural NLG system that has an identifiable surface realiser can be found in [51]. However, its surface realiser is not data driven. Instead, it is fed templates with abstract representations of verb phrases ans noun phrases. These templates are produced by the preceding modules. It's architecture belongs to category A6 and it will be introduced in the following section.

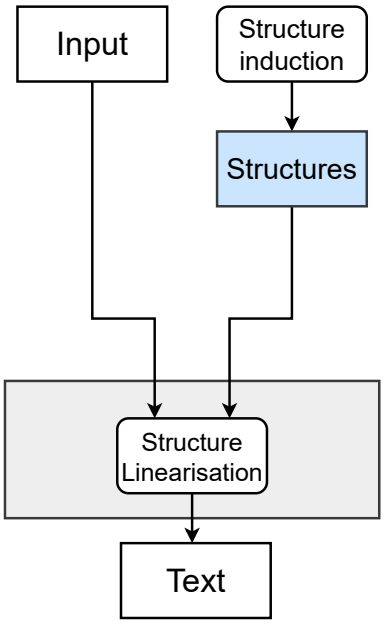


Figure 5.8: Surface realiser architecture category AC6. We use the grey box with sharp corners to denote the surface realiser and a blue box with sharp corners to denote that the structures are captured using a template that may be grammar-infused.

Architecture category 6 (AC6): simple declarative structures The AC6 architecture category is used by a few systems. Their realisers are responsible for a single task, i.e., the linearisation of structures. For instance, Castro Ferreira's [51] realiser takes a template with underspecified verb and noun phrases. These templates look like the one shown in Listing 5.1.

Listing 5.1: Example of a template found in Castro Ferreira et al.'s pipeline neural system

```
1 Massimo Drago VP[...] play for DT[...] the club
2 SSD Potenza Calcio and his own club VP[...] be
3 Calcio Catania . He VP[...] be currently VP[...]
4 manage AC Cesena .
```

The realiser then uses rules[4] to resolve the verb and noun phrase values oin order to generate the final string. Another system whose realiser abides by the organisation seen in AC6 can be seen in Gubbala et al.'s [102] work[5]. Unlike Castro Ferreira et al.'s neural pipeline system, this one is not preceded by neural modules and it uses RosaeNLG to generate text.

[4] https://github.com/ThiagoCF05/DeepNLG/blob/master/realization.py

[5] https://github.com/Alihussainladiwala/Citizen-Friendly-Report-of-diversitydatakids.org/

5.4 Limitations for Nguni languages

The architectures that belong to the six categories are not appropriate for NCB languages. Architectures that belong to categories AC1/2 are often used by domain-independent realisers that require the creation of an input specification for purposes of limiting the realiser's semantic space, since they make tactical decisions the responsibility of the realiser. As a consequence, they require detailed input, hence, making them difficult to easily accommodate in a new system [93, pg80]. This is affirmed by the low adoption of large-scale grammar-based realisers that use architectures AC1/2 (e.g., KPML and AlethGen/GL) in modern NLG systems. These control issues and the lack of large coverage grammars for NCB languages means they are not appropriate.

Contemporary NLG systems use the architectures that belong to categories AC3/4/5/6. Most domain specific systems that are designed for well-resourced languages make use of AC4 architectures, especially when reliability is of concern. Recently, there has been an uptake in NLG systems that follow AC5 architectures for such languages. However, such systems are rarely used in areas where reliability is important since they sometimes exhibit hallucinations, incoherency, and degenerative repetitions [110]. AC3 architectures have been used by a number of NLG systems that support low-resourced languages, especially African languages. These four architecture categories have a number of strengths:

- It has been demonstrated that it is possible to bootstrap an existing system for a related language if they follow AC3 (see [44]).
- AC4 moves tactical decisions out of the realiser and leaves minimal tasks; hence, does not require restrictive input specifications.
- Systems that abide by AC5 make use of automated means for mapping the input to the output; hence, there is no need for hand-crafted modules whose construction may be time-consuming.
- AC6 yields small-scale surface realisers that are supposed to be easy to use (e.g., [102])

Despite the aforementioned benefits, architectures that belong in categories AC3/4/5 are also unsuitable for NCB languages with respect to maintainability at the time of writing for the following reasons:

- AC4 relies solely on grammar engines for structure realisation, hence, are unsuitable for languages that are low low-resourced. IsiXhosa and isiZulu are low-resourced and have limited up-to-date grammar literature; hence, it is not feasible to develop large coverage grammar engines. Architectures that belong to this category would only be viable and appropriate if a full-scale grammar engine existed.
- The use of architectures that belong to category AC5 would only appropriate for languages and domains where there exists parallel data to text resources to train statistical/neural models. However, since isiXhosa and isiZulu do not fit that description then it is not appropriate. This is demonstrated by the sizes of datasets used to train statistical and neural systems, as given in Section 2.4.1.3. Even if the relevant data were available, systems that make use of this architecture, especially when they use neural approaches, may not be appropriate in certain circumstances since they are prone to generating text that is not associated with the input (i.e., hallucinate) [243], drop relevant information [232], and needlessly repeat text at the end of sentences [171].

The only promising architecture categories for the languages in question are AC3/AC6. For instance, all existing NLG systems for Nguni languages have used architectures that belong to category AC3. The only limitation with category AC3 is that it does not yield maintainable and reusable software as it tightly-couples the linearisation rules with the actual templates and makes tactical decisions the responsibility of the realiser. For instance, consider the algorithm[6] implementation for verbalising negated simple existential quantification given in Listing 5.2.

Listing 5.2: Python implementation for verbalising simple existential quantification in isiZulu (Source: [138])

```
def nexist_zu(sub,op,super):
    nc1m = find_nc(sub)
    nc2m = find_nc(super)
    nc2 = strip_m(nc2m)
    pl = plural_zu(sub,nc1m)
    ncp = look_ncp(nc1m)
    qca = look_qca(ncp)
    rc = look_relc(nc2)
    qc = look_qce(nc2)
    rt = find_rt(op)
    negsc = look_negsc(ncp)
```

[6] https://github.com/mkeet/GENIproject

```
12      if rt[0] in 'aeiou':
13          negconjrt = negsc_vowel_vroot(rt,negsc)
14      else:
15          negconjrt = negsc + rt
16      return qca + ' ' + pl + ' ' + negconjrt + 'i' + ' ' + super + ' ' + rc
            + qc + 'dwa'
```

The algorithm implementation given in Listing 5.2 is one of the implementations of the algorithms described in [136]. When the verbaliser is given the input $Person \sqsubseteq \neg\exists achieve.Task$, it identifies that the relevant lexical items to use are *umuntu* 'person', *feza* 'achieve', and *umsebenzi onqunyiwe* 'task' for the various elements, and invokes nexist_zu (line 1). The above implementation starts by identifying and processing the noun classes of the nouns associated with the sub and super concepts (lines 2-4). It then pluralises the sub's noun and identifies the plural's noun class (line 5-6). Once that information is obtained, it makes use of the retrieved noun classes information to identify the relevant quantitative and relative concords (lines 7-9). Then, a negated verb is formed for object property (lines 10-15). Once all the above elements are identified and formed, the algorithm them combines them to form the sentence *Bonke abantu abafezi umsebenzi onquyiwe oyedwa* 'All humans do not achieve some task' (line 16). In this scenario, the grammar-infused template and the linearisation rules used to generate such text do not exist separately. In addition, the decision to choose an appropriate template given some input is also the responsibility of the realiser. From a re-usability perspective, this means that one cannot isolate the templates and consider them for re-use.

Technically, systems that follow this architecture can be re-written such that they abide by architecture A6. In such a redesign, the templates could be captured via a separate language (à la RosaeNLG's pug templates) and the choosing of an appropriate template is done by a separate module. For instance, the above algorithm can be split into a template and linearisation algorithm. The aforementioned hypothetical template would look like the one given in Listing 5.3.

Listing 5.3: Hypothetical RosaeNLG template for simple existential quantification in isiZulu

```
1 concateList
2       | +value('onke', {nc: getNC(sub)})
3       | +value(sub, {number: PLURAL})
4       | +value(op, {marker: NEGATIVE})
5       | #{super}
6       | +value('odwa', {nc: getNC(super)})
```

In the template above, concateList is hypothetical extension of the pugjs[7] template language. Its purpose is to order plain-text elements and may intelligently introduce spaces between some special elements. We also make use of the value mixin from RosaeNLG (lines 2-4, 6) with parameters that are currently not supported. Line 1 can be understood as invoking the value mixin in order for the template engine to generate the appropriate form of *onke* 'all' depending on the noun class the noun associated with the *sub* element. The rules for linearising the template can be moved to separate and invoked in the manner shown in Listing 5.4.

Listing 5.4: Hypothetical JavaScript code for using a RosaeNLG template to verbalise simple existential quantification in isiZulu

```
1  const rosaenlgPug = require('rosaenlg');
2
3  function verbalise(axiom):
4      ...
5      if nExistAxiomMustBeUsed:
6          vals = getLexicalValues(axioms)
7          nexist_zu(vals[0], vals[1], vals[1])
8
9  function nexist_zu(someSubVar, someOpVar, someSuperVar):
10     return lineariseTemplate(
11     'template.pug',
12     {
13         language: 'zu',
14         sub: someSubVar,
15         op: someOpVar,
16         super: someSuperVar
17     });
18
19 function lineariseTemplate(templateName, params)
20         return rosaenlgPug.renderFile(
21           templateName,
22           params);
```

In the above hypothetical snippet, there is a function for choosing the appropriate template when given an axiom (lines 3-7), a function for retrieving the template from a file and passing the lexical items for axiom's elements (lines 9-17), and a reusable function for linearising templates that makes use of RosaeNLG's engine (lines 19-22). The use of a declarative template language results in templates that are separate from the linearisation rules. As such, they can be reused without the need to recreate the relevant rules.

While the use of a declarative template language increases the opportunity for resource reuse, it also introduces a challenge with respect to constraint checking. The

[7] https://pugjs.org/language/

simple templates and grammar-infused templates that were introduced in Chapter 2 have a number of constraints by definition. For instance, in both kinds of templates, words and slots must be ordered and there must be at least one slot. In addition, one is free to specialise the two concepts by introducing additional constraints. However, since the template file will only specify the 'what' and not the 'how' in the hypothetical template given in Listing 5.3, this opens up the possibility of reusing invalid templates and that may have an impact of the viability of template re-use. This issue could be solved through the following methods:

- Best practice guidelines: one can collect pitfalls seen in template reuse over an extended period of time and create methodologies for template-reuse that ensure that NLG engineers avoid those pitfalls. The disadvantage with this approach is that it can only lead to a solution after one has spent an extended amount of time observing pitfalls;
- Safeguards in system architecture: another approach is to make use of the presented analysis of existing NLG architectures to introduce changes that ensure that the architecture has robust means for template constraint violation detection.

In the next section, we present our work on knowledge-guided architecture. It has safeguards for constraint violations in templates (and possibly other methods used for structure encoding).

5.5 Knowledge guided architecture

We develop the architecture by first specifying requirements, create an architecture that meets the requirements, and then evaluate the result through a manual feature comparison.

Requirements The requirements phase focuses on determining the high level characteristics that should be in the architecture. These requirements are based on observations we have made on the evolution of realiser architectures in NLG, as discussed in the previous section, and the needs of the languages in question. The high-level design requirements are as follows:

- The sentential structure must be encoded via templates, which may be grammar-infused, in order to support low-resourced languages. When necessary, it must

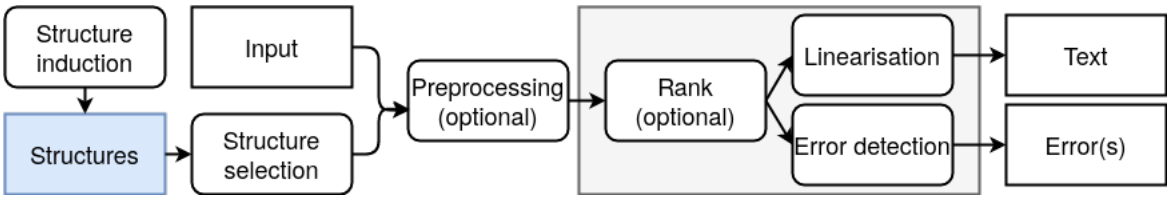


Figure 5.9: Functional view of the knowledge-guided architecture to generate text from grammar-infused templates.

be possible to enrich the grammar-infused templates with task/context specific grammatical knowledge.

- Knowledge regarding grammar-infused templates must be incorporated via application ontologies as such a feature would allow us to be able to use semantic technologies for purposes of template creation, automated reasoning to detect inconsistencies, and comparison of template features. At first glance, it may seem like the approach of capturing templates separate from a surface realiser, but it is not. The use of grammar-infused templates requires a principled approach to encoding the necessary grammatical and template knowledge. Just like there are formal theories for specifying grammars, a theory of templates is required in order to make the combination sound.
- Tactical decisions should not be the responsibility of the surface realiser. Similar to the proposal by SimpleNLG [94], tactical decisions should be left to the discretion of each NLG engineer. Doing so avoids the control issues and restrictive input formalisms found in dated wide-coverage tactical generators.
- Maintainability must be a key feature guiding the design of the architecture since template-based systems have been criticized in the past for being difficult to maintain. In particular, we are interested in modularity, componential reusability, and analysability.
- Statistical models should be usable as a means for the creation and selection of templates when generating text. Moreover, the component for selecting a template given some input must not be necessarily data-driven since the language for which one is building an NLG system may not have sufficient corpora to build a data-driven template selector. It must be possible to also make use of the grammatical features afforded by grammar-infused templates to rank possible templates (cf. only using surface text).

Architecture creation Based on the above requirements and our analysis of existing architectures, we have created the architecture shown in Figure 5.9. The *tactical*

decisions are moved out of the realiser similar to AC4. *Structure creation* is achieved via templates, possibly grammar-infused. Moreover, for a faithful implementation of this architecture, template *knowledge* (i.e., concepts, relationships, and constraints) must be formalised in an ontology. Lastly, there is no prescription for the methods that can be used for structure induction so rules or *statistical models* can be used. Further, when generating text using an implementation of this architecture, there might be multiple candidate grammar-infused templates, as a result of needing variety for text to be generated. The *ranking* component selects an appropriate template from the candidates, which can be rule-based or data-driven. The selected template and its associated input are then passed to the linearisation module for slot value insertion, application of the necessary grammar rules, and finally producing surface text.

The *preprocessing* and *ranking* modules are introduced and placed in a location that differs from the architectures presented in Table 5.1 and categorised in Table 5.2. This is done to satisfy the requirements pertaining to adding task-specific grammatical information and using it to rank the possible templates. To demonstrate the usefulness of such features, we use Braun, Reiter, and Siddharthan's [37] SaferDrive system. The system was originally created to generate reports to make drivers aware and reduce their negative driving habits. One could extend the system for use by insurance companies so that they may offer their version of reports to drivers or to report to actuarials who set prices depending on driver behaviour. The same templates could be used, but the grammatical voice would be different. For instance, templates for when a driver accesses the reports directly can be annotated in the active voice through the pre-processing module, but for the insurance company it would be annotated with passive voice. Consequently, when given multiple templates for verbalising the message, say, drivingPeriod(distance, time), then the ranking module would select the appropriate template given the context; e.g., when generating text for the driver, it may generate "You drove 390 miles in 10 hours and 50 minutes during the last week" [37, pg576] and for the insurance company it would then be "390 miles were driven in 10 hours and 50 minutes during the last week by X" where X is the name of the driver.

5.6 Architecture maintainability

In this section, we advance an *a priori* justification for maintainability. In particular, we are interested in the (ab/pre)sence following features: support for economical

Table 5.3: Comparison of the new architecture to other architectures. Abbreviations: $\mathcal{C}$ = Computational grammar rule(s) reuse, $\mathcal{O}$ = Ontology formalised concepts and relationships, $\mathcal{S}$ = Separation of surface realisation tasks. Legend: Green = supported, Black = not supported, and Dash = not applicable.

Architecture	**Category**	$\mathcal{C}$	$\mathcal{O}$	$\mathcal{S}$
Proposed architecture	-	●	●	●
0, 1, 3	AC1	●	-	●
2	AC1	●	-	●
4	AC2	●	-	●
5	AC2	●	-	●
9, 10	AC4	●	-	●
11	AC5	-	-	-
7	A5	●	●	●
8, 6	AC3	●	●	●
12	AC6	●	●	●

resource use (componential re-usability), having well defined template concepts and relationships between the concepts (analysability), and ensures the separation of surface realisation tasks (modularity). We do not include the other two maintainability characteristics of [38], namely, modifiability and testability, since they require *a posteriori* justification and there is no implementation of the architecture that has been used over an extended period. Henceforth, we use the symbol $\mathcal{C}$ to refer to our operationalisation of *C*omponential re-usability, $\mathcal{O}$ to analysability of the knowledge formalised in an *O*ntology, and $\mathcal{S}$ to modularisation of the *S*urface realisation tasks.

Table 5.3 shows each architecture's support of the maintainability features. We demonstrate them with the isiZulu verbaliser [138]. Analysis of the verbaliser's architecture (Arch. id. 7, category AC5) shows that it supports reusable grammar rules for pluralisation and other rules can be added in the same fashion (yes to $\mathcal{C}$). However, the algorithms used to generate text encode the the selection of a specific template for each quantifier and their corresponding linearisation rules (no to $\mathcal{S}$). The verbaliser has a clear notion of what template concepts are allowed and how they relate to each other, even though the constraints are not explicitly declared. In addition, the permissible template concepts can be extended when needed in a similar fashion to how Keet and Khumalo [134] extended the isiZulu patterns to support part-whole relations. Unlike Keet and Khumalo [134], extension is not done in an *ad hoc* manner and our templates are formalised via an ontology (yes to $\mathcal{O}$).

Our architecture combines the strengths of AC1-6, as stated via our architecture requirements, and supports the most maintainability features. It is the best option for NCB languages since it supports the specified requirements and most of the maintainability requirements (our operationalisation of [38]'s maintainability characteristics to ensure that they are sensitive to the needs of NCB languages). It is not necessarily the best choice for other languages, especially well-resourced languages, since it enforces the use of templates.

It differs from architectures found in category AC4, the most popular architecture category in extant systems, since it does not rely only on a grammar engine but places templates at the center. It also introduces an ontology-based specification of templates. This allows NLG systems to use automated reasoners to detect inconsistencies and also support interoperability in template-based systems.

It is not necessarily the best choice for other languages, especially resourced languages, since it enforces the use of templates. In cases where one wants to avoid the use of templates then architectures belonging to AC4/5 may be best. This is subject to availability of a grammar engine (AC4) and the toleration of degradation in textual coherence and correctness associated with data-driven NLG models (AC4). Nonetheless, our architecture may still be used for such languages as it has no prescript regarding structure induction. This is meant to tackle the challenge of template creation and maintenance. As such, one can pursue data-driven techniques in the same vein as [144] for template creation.

As can be seen in Table 5.3, it is not possible to mix and match features from architectures in each category to satisfy all the maintainability features. For instance, using A1 as an example, one cannot combine the best best features of architectures 0, 1, 3, and 2 in order to satisfy $\mathcal{C}$, $\mathcal{O}$, and $\mathcal{S}$. This is primarily because none of the architectures support the use of ontologies to capture template knowledge.

5.7 Value of template ontology

We will demonstrate the utility of the ontology for grammar-infused templates as a means of facilitating the detection of logical inconsistencies in templates, sharing, and the comparison of templates cf. *ad hoc* approaches. We will use demonstrative soccer templates and slot fillers in the next section and Sanby et al.'s two systems to demonstrate these benefits.

5.7.1 Inconsistencies

The benefit of using an ontology cf. just an XML schema definition language (or similar language) is automated reasoning. That is, the knowledge in the ontology (the TBox) can be combined with the templates (encoded as instances in the ABox) to create a knowledge graph that then can be sent to an automated reasoner (e.g., HermiT OWL Reasoner [96]). Consider the following templates:

```
(a) Slot[player] Phrase[scored the third goal].
(b) Phrase[The goal at the 20th minute was]  AltPhrasing(Phrase([
   not offside]), Word([onside])).
(c) Phrase[The winning goal was the result of a header by] Slot[
   player].
(d) Key[player], Value[Aardvarkis]
```

The templates above could be used to generate messages associated with the meaning representation goalEvent(scorer, time, goalLineStatus, goalNumber) in some system(s). In such a scenario, the ontology is used to declare the concepts to be used when creating a template (à la schema). In this instance, we declare the concepts as Slot, Phrase, Value, and AltPhrasing. The ontology can also be used to specify constraints. For instance, to specify that templates must have at least one slot, i.e, `Template` $\sqsubseteq \geqslant 1$ `hasPart.Slot`. It is the responsibility of the NLG engineer to create the templates such as (a), (b), and (c) together with the key-value pair (d) to be inserted in the templates. The templates can then be serialised using any number of formats (e.g., RDF/XML) and stored in a file.

The templates and their associated ontology, they can be loaded into the reasoner and checked if they are consistent. If the ontology includes the aforementioned axiom restricting the number of slots in a template, then it will detect that template (b) is not a valid template because it does not contain any slots. For simple templates, it may seem like this is an insignificant benefit since it has to be balanced against the work required to create a template ontology. However, the ontology presented in Chapter 4 supports simple templates; hence, there is no need to create a new template ontology. Moreover, the ability to detect inconsistencies is vital for the grammar-infused templates that are used to generate morphologically rich languages. In such languages, there is a likelihood of making errors when defining their polymorphic words.

Overall, it may seem like the ability to detect that template (b) is not valid, in the given example, is not impressive. However, one be aware that when using

grammar-infused templates that have numerous embedded Computational Grammar Rules (CGRs) then the number of restrictions found in the templates also increases. As such, the opportunities for making mistakes also increase. The value of detecting violations must be understood within that broader context.

5.7.2 Template comparison and reuse

Use of an ontology also facilitates sharing and comparing templates. For instance, the two systems found in Sanby, Todd, and Keet [249] both verbalise Web Ontology Language (OWL)'s disjoint classes axioms. One system does so using an Extensible Markup Language (XML) template and another uses a Grammatical Framework (GF) concrete grammar rule. We will demonstrate using Task ontology for CNL Templates (ToCT), how they are isomorphic despite the difference in the language they use.

First, we begin by explaining the elements of the two templates; the XML template's *Text* object and the strings in the GF grammar rule are used to capture fixed texts only. The difference between the approaches can be seen in line 4 of both listings when they capture the fixed string *nie* 'no'. Reusing the terminology from our soccer example given in Section 5.7.1, it should be clear that XML and GF features serve the same purpose as the *Phrase* concept. The XML template's *Object* and GF's variables x and y are used as place-holders for different values that can be used. For instance, line 3 in both listings marks the position that will be occupied by the name associated with first the class that participates in the Disjoint axiom. If the input is the axiom Disjoint(Animal Computer), then the position might be occupied by the noun 'animal'.

Listing 5.5: XML template for verbalising OWL's Disjoint axioms

```
1 <Constraint type="Disjoint">
2 <Text>'n</Text>
3 <Object index="0"/>
4 <Text>is nie 'n</Text>
5 <Object index="1"/>
6 <Text>nie</Text>
7 </Constraint>
```

Listing 5.6: GF template for verbalising OWL's Disjoint axioms

```
1 DisjointClasses x y = { s=
2 "'n "
3 ++x.s++
4 "is nie 'n"
5 ++y.s++
6 "nie"
7 };
```

Analysis of the concepts used in the two types of templates shows that they have the same meaning and function, as demonstrated in Figure 5.10. Specifically, the XML's *Object* and GF's variables both refer to ToCT's *Slot* concept. The *Text* and GF's

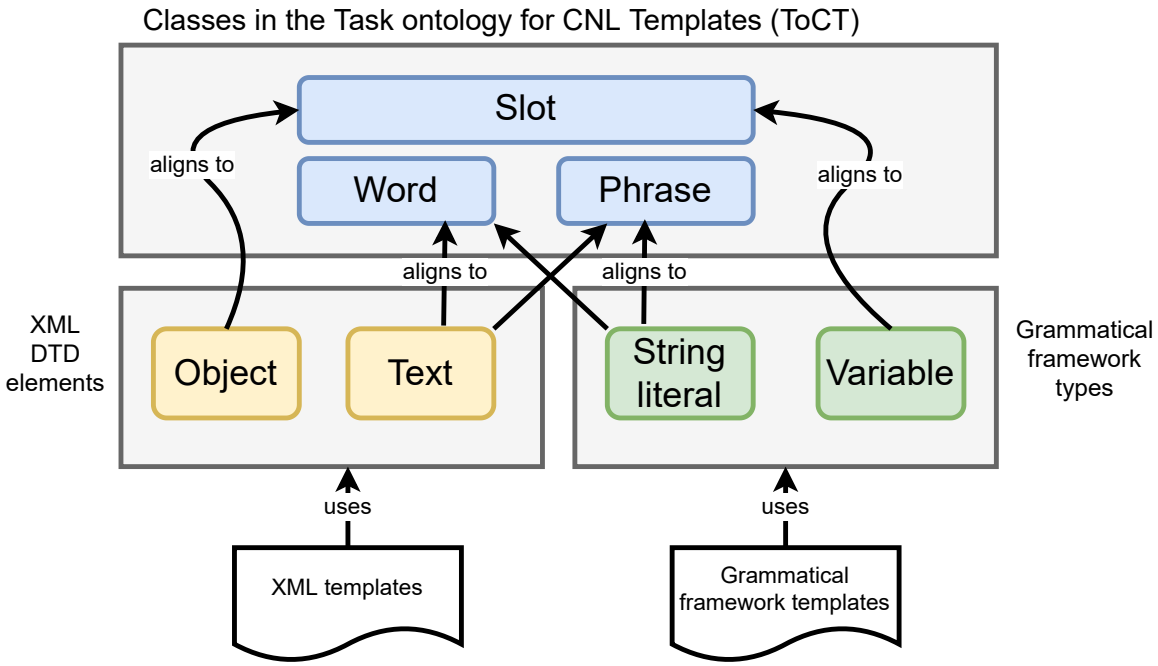


Figure 5.10: Representation of the concepts used in the GF and XML templates and how they relate to each other. The disparate specifications, in XML's document type definition (DTD) and Grammatical framework (GF), are aligned to classes found in our ontology, ToCT.

strings literals are used to refer to ToCT's Unimorphic word and a special case of the Phrase concept.

ToCT's common vocabulary shows that when the bells and whistles are removed from the above listings, we are only left with equivalent fixed phrases and slots. Thus, the two verbalisers can share the template, map it to their preferred serialisation, and avoid resource duplication.

While one could develop an *ad hoc* meta-model for achieving this for Sanby, Todd, and Keet's [249] templates, the reliance on a ontology is a better approach because of the benefits outlined in the previous section and the ontology may be used by a wider set of template-based systems, especially if it abides by the Findable, Accessible, Interoperable, Reusable (FAIR) principles similar to the one presented in Chapter 4. These capabilities are not possible in the other existing NLG/surface realiser architectures because they either do not use templates or use *ad hoc* template specifications.

5.8 Summary

In this chapter, we have presented a novel surface realiser architecture that moves tactical decisions out of the realiser, relies on templates (possibly grammar-infused) for encoding sentential structure, enforces the use of an ontology to formalise the the templates' concepts, relations, and associated constraints. It also supports candidate ranking via a module that may be data-driven. Unlike existing architectures, especially those used by NCB language generation systems, our architecture is designed to produce surface realisers that are easy to maintain since it supports our operationalisation of component re-usability, analysability, and modularity.

In the following Chapter, we turn to the evaluation of our ontology (i.e., ToCT) and the architecture via an implementation of text generation systems for isiZulu and isiXhosa.

Chapter 6

Evaluation

We now turn to the evaluation of the architecture and its respective components for data and knowledge to text generation systems.

The purpose of the evaluation is to demonstrate that the two created Natural Language Generation (NLG) systems can capture complex templates and their generated texts are well-formed. In addition, the generated texts are perceived favourably by speakers of isiZulu and isiXhosa. This is demonstrated in scenarios where the templates are several polymorphic words and make use of the Task ontology for CNL Templates (ToCT).

The rest of the chapter is structured such that Section 6.1 presents an overview of the evaluation strategy taken, Section 6.2 presents implementation of grammar engines for isiXhosa and isiZulu, the implementation of the linearisation module, and the error detection module, Section 6.3 presents the the isiXhosa weather forecast generator and the quality of its output, Section 6.4 presents the design the templates used for the isiZulu text generator and the quality of generated questions, and Section 6.3 presents a discussion of the two systems with particular emphasis on ToCT's scope and quality of the generated text.

An early version of the work presented in Section 6.4 was published at the 3rd International Workshop on Natural Language Generation from the Semantic Web [169]. We have updated it to improve readability and redesigned its architecture to use the one presented in Chapter 5.

6.1 Evaluation strategy

Our evaluation uses a weather forecast text generator and ontology verbaliser as our main instruments. We design a weather forecast generator for isiXhosa and an isiZulu text generator used for verbalising an ontology's TBox as validation questions. We use two systems that belong to the data-to-text and knowledge-to-text categories to demonstrate that the architecture, and its components, is capable of generating text for both kinds of NLG system categories. Building such systems requires a surface realiser. Since none currently exists for isiXhosa and isiZulu, before evaluating our work using the aforementioned systems, we first present proof-of-concept Java implementation of the various modules and artefacts needed for a surface realiser, as listed in our architecture that was presented in Chapter 5.

In order to create an surface realiser that abides by the architecture, a number of modules and resources are required. They are an ontology for specifying the sentential structures (in casu templates), an optional method for ranking templates and slot fillers, a module responsible for linearising sentential structures, and an error detection module. We will only present proof-of-concept implementations of grammar engines, linearisation modules, and error detection modules. We use ToCT, first presented in Chapter 4, to capture sentential structure and since the ranking module is optional and not the focus of this thesis, we omit it.

Our evaluation mode is subjective [24] and its purpose is to determine goodness and correctness of the text produced by "the implementation [of the described components, first described in Section 2, with respect to] grammatical features" [36, pg418]. We take a different approach Braun et al. [36] in that we do not compare generated texts to a corpus to determine the scope of the artefacts and resources. We also do not make use of the strategy seen in the Surface Realisation Shared Task [23] where one compares the surface realiser's output to some corpus using automated metrics such as BLEU [218]. If we were to follow aforementioned approaches then we would need to compare our generated text to a large and diverse "gold-standard corpus" [244]. Such a corpus would allow the identification of variations, their linguistic properties, and the similarity of our generated text to the 'gold-standard' corpus. This is important since the proposed artefacts are created to cover a variety of different domains. However, there are no large (data/knowledge)-to-text datasets for the languages that suitable for the task and creating them is not practical.

In order to make Braun et al.'s [36] evaluation approach more suitable for purposes, we had considered downloading a large monolingual dataset (e.g., the aggregated datasets presented in [240]), extracting templates and capturing them using ToCT, creating mock data/knowledge to act as the input for each template, generating text, then comparing the output to the original monolingual corpus. However, this was not done since it is also not practical as the templates and mock input would still need to be created manually. Unlike Braun et al.'s [36] test structures, our templates cannot be created semi-automatically since there are annotations associated with the aforementioned datasets or any other datasets.

As an alternative to Braun et al.'s [36] approach, we settled with documenting the linguistic phenomena covered by the proof-of-concept implementations, using unit testing for internal verification/validation of the correctness of the captured phenomena, and relying on the subjective human judgements of the generated texts to determine the correctness of the various linguistic phenomena.

6.2 Surface realiser implementation

We now turn to the implementation of the language-specific grammar engines, the linearisation module, and error detection module. The components are used to create the surface realiser, which can be used as part of a larger NLG system to generate well-formed natural language sentences when given templates and slot fillers/values. We will discuss the features covered by the isiZulu and isiXhosa grammar engines in Section 6.2.1. We will also present the internal validation of the algorithm implementation.

6.2.1 Grammar engines

We created Java proof-of-concept implementations of grammar engines for isiZulu and isiXhosa. The features that they currently support are as follows:

- Concord resolution: These rules retrieve the value of each type of concords based on the provided noun class. These rules are implemented for both languages and the concord values are taken from Bourquin [34].
- Copula resolution: These rules retrieve the value of a copula using a noun class and, sometimes, the morpheme that is to follow the copula. These are currently implemented only in the isiZulu engine. The current rules return *w* for noun class 11, return *ng* if the next affix begins with *u-*, *o-*, and *a-*, and returns *y* in

all other cases. They are not implemented for isiXhosa because the chosen use case did not require them. We prioritised rules to include in each engine based on the considered use cases.

- Locative resolution: These rules retrieve the value of the locative using a noun class and the morpheme that is to follow it. These are currently implemented only in the isiZulu engine. The current implementation returns *ku-* for noun class 1, 2, 1a, and 2a and returns *e-* for all other classes. These are also are not implemented for isiXhosa because the chosen use case did not require them.
- Noun class resolution: This is an abstract interface for models/rules to be used to determine the noun class of a specific string. In the isiZulu verbaliser described in Section 6.4, a dictionary-based implementation is introduced.
- Morphophonological alternation: These rules are the various phonological conditioning rules that were described in Section 2.1.3. These rules are implemented for both languages.

The design of the engines took advantage of Java's polymorphism for the two languages to rely on the same interface. This approach creates a single interface for the linearisation module for all Nguni languages and avoids the creation of multiple interfaces even though a single implementation suffices for all languages.

The created engines do not have comprehensive coverage, especially when compared to SimpleNLG's [94] coverage for English. Nonetheless, they are significant to Nguni languages since they currently support linguistic processes at the word level. At the time of writing, the limited coverage is not a limiting factor since the engines are created as proof-of-concept auxiliary tools to support the text generators to be introduced in Sections 6.3 and 6.4.

The proof-of-concept implementations (i.e., the grammar engines, error detection module, and linearisation module) can be downloaded from `https://github.com/AdeebNqo/NguniTextGeneration`.

6.2.2 Error detection

This module corresponds to the 'Error detection' module introduced in Section 5.5. Its purpose is to identify logical inconsistencies in the templates and allow one to filter out low quality templates.

We used the Java OWL-API[1] library to read ToCT and its Tbox is then combined with a template specified via ToCT, representing the ABox. The resulting knowledge graph is used to detect inconsistencies via the FaCT++ 1.6.5 [270] automated reasoner.

6.2.3 Linearisation

To generate text from templates specified using ToCT, we create the linearisation algorithm that is presented as Algorithm 1. It is capable of linearising such templates for any domain.

Algorithm 1 Algorithm for linearising a morphologically-enhanced simple template

Input T: template
SV: slot fillers
CM: language-specific concord mapping
MR: language-specific morphophonology rules

1: Initialise s to empty string
2: **for** $lexItem \in$ getOrderedWords(T) **do**
3: **if** $lexItem \doteq Slot$ **then**
4: $slotValue =$ getSlotValue$(lexItem, SV)$
5: insertValueInSlot$(lexItem, slotValue)$
6: **if** $lexItem \doteq PolymorphicWord$ **then**
7: $govs =$ getGovernorValues$(lexItem)$
8: fillInConcordValue$(lexItem, govs, CM)$
9: $wordValue =$ getWordValue$(lexItem, MR)$
10: insertValueInWord$(lexItem, wordValue)$
11: $s \leftarrow s+$ getValue$(lexItem)$
12: **return** s

The algorithm works by creating an empty sentence and iteratively appending the final form of each template fragment. Specifically, the main goal of the algorithm is to insert the input slot fillers/values into their corresponding slots (lines 5 and 10). In the algorithm, we use the operator $\doteq$ (lines 3 and 6) to mean "is of type" (i.e. $x \doteq y$ means that x is of type y). Filling in those slots sometimes relies on the construction of polymorphic words. We form polymorphic words by retrieving their governor(s) and using them to retrieve the polymorphic word's concord values (lines 6-10). Since concord values differ across languages, this retrieval process relies on a language-specific mapping of the various concord values and their types (CM in line 8). Once they are retrieved and filled into a polymorphic word, the language-specific

[1] https://github.com/owlcs/owlapi/

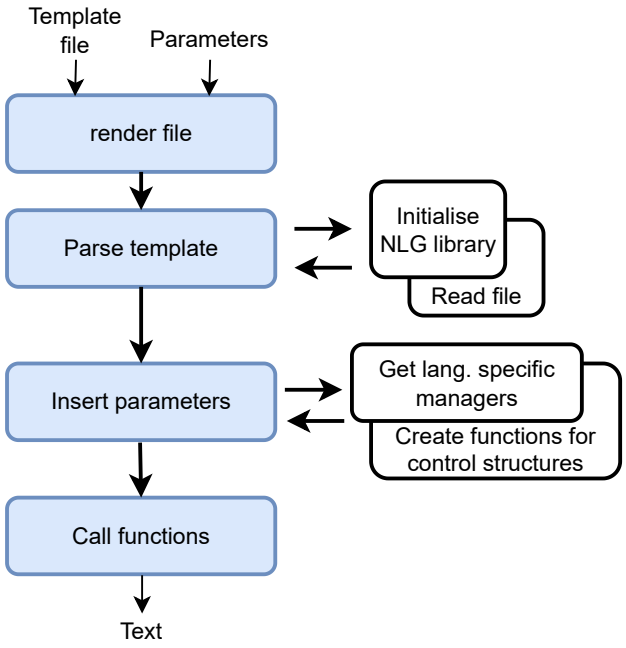


Figure 6.1: The process used by RosaeNLG to generate text when given a template file name and some associated parameters. Abbreviation(s): Language = lang.

Computational Grammar Rules (CGRs) responsible for phonological conditioning are applied to create a well-formed word (line 9). The value of the word, slot, or other template portion is then appended to the sentence to be generated (line 11). Finally, the linearised sentence is produced (line 12).

The algorithm works in a way that is similar to RosaeNLG. The latter takes a template and some parameters, some of which will be inserted into slots, following the process demonstrated in Figure 6.1. RosaeNLG's engine starts by initialising the library that has language rules and parses the template file. It then inserts the relevant values passed as parameters into slots. Specifically, this is done by first retrieving the language specific modules that capture various linguistic features and associating JavaScript functions to the control structures that were specified in the parsed templates. Finally, the associated functions are all activated thereby linearising the template to generate the final text. Our algorithm differs from RosaeNLG in that it not tied to a fixed set of grammar rules, it includes the notions of concords to make it suitable for Niger-Congo B languages, and it is documented vs. RosaeNLG's undocumented Typescript-only implementation.

Table 6.1: Templates to be used for testing the linearisation algorithm for isiXhosa and isiZulu

Dependency	Template	Translation
Possessive	⟨noun⟩ ⟨*C*⟩madoda	⟨noun⟩ of men
Subjectival	⟨*C*⟩onke ⟨noun⟩	all ⟨noun⟩
Subjectival and objectival	⟨$noun_1$⟩ ⟨C_1⟩ya⟨C_2⟩hambisa ⟨$noun_2$⟩	⟨$noun_1$⟩ makes ⟨$noun_2$⟩ leave
Phi/Mbi (Xh)	⟨*noun*⟩ ⟨*C*⟩mbi	another kind of ⟨*noun*⟩
Long adjectival	⟨*noun*⟩ ⟨*C*⟩dala	a ⟨*noun*⟩ that is old
Short adjectival (Xh)	⟨*noun*⟩ ⟨*C*⟩dala	old ⟨*noun*⟩
Relative	⟨*noun*⟩ ⟨*C*⟩hambileyo	⟨*noun*⟩ that has left
Enumerative	⟨*C*⟩nye ⟨*noun*⟩	one ⟨*noun*⟩

6.2.3.1 Validation

To validate our linearisation algorithm and its implementation, we created one template for each type of dependency documented by [34, 192] of at most four words. We limited the number to four so as to ensure that we can include the necessary parts of speech and have succinct templates, but avoid introducing unintended errors. For instance, for the enumerative dependency type, we used the following template where C denotes a concord.

⟨*C*⟩`nye` ⟨*noun*⟩

The first item is used to form words of the kind *umnye* 'one'. The complete list of templates, and their English approximation, is given in Table 6.1.

We then randomly selected two noun classes (one singular and one plural) and selected one noun per class for each language to use as slot fillers for the created templates. We linearised the templates and checked if the resulting text is the same as the expected correct text. In our random selection of nouns, we obtained noun class 10 and 1 for isiXhosa and then used the words "$izinja_{10}$" and "$umntu_1$". We obtained noun class 4 and 7 for isiZulu and then used the words "$imifula_4$" and "$isithombe_7$". All templates generated correct text.

Having created the various resources, we now turn to discuss the text generation systems and the evaluation of their output.

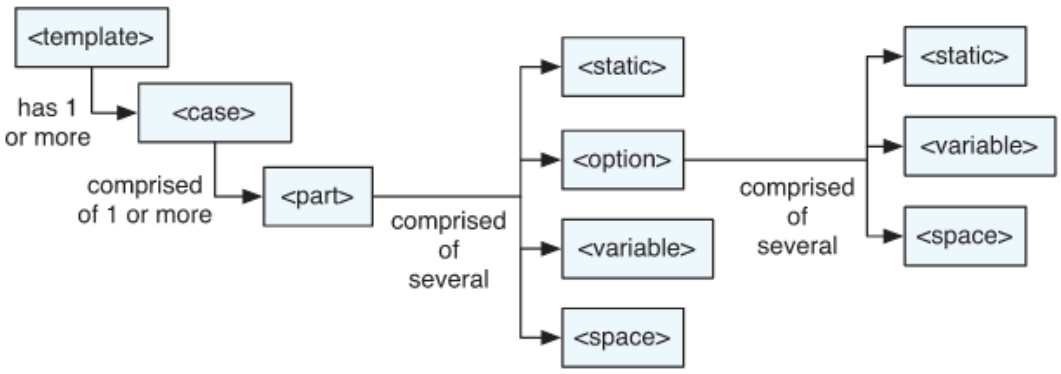


Figure 6.2: Template schema used in the GALiWeather system (Source: [234])

6.3 IsiXhosa GALiWeather

The only existing weather corpus in isiXhosa is too small to be used to extract reliable templates. Building a new large corpus is impractical due to the unavailability of such data [167]. As such, we have opted for adapting an existing multilingual weather forecast generator to isiXhosa. Since weather forecast generation is a popular application domain within NLG, we considered several systems' corpus for adaptation to isiXhosa. We searched for a weather corpus designed for general everyday in-land use (cf. pollen forecasts [271] or marine weather forecasts [259]) using the list of weather forecast generators found in [167] as a starting point. We eventually decided on using GALiWeather [234] since it was created for generating forecasts for everyday in-land use and its templates were accessible. The templates, and their corresponding NLG system, were created for producing weather forecasts for various municipalities in Spain.

6.3.1 Methods and Materials

We downloaded and translated the English GALiWeather templates [234] to isiXhosa.

A single English GALiWeather template may have multiple forms and fragments with multiple forms – since its schema is such that the template allows multiple cases and a template can have multiple optional text segments (see Figure 6.2).

We demonstrate GaliWeather's templates using the one used to express changes in temperate:

```
1 The temperatures will be
```

```
2       [norT] for this period of the year,
3       [minT] for the minimums and [maxT] for the maximums
4       compared to the expected for this time of the year,
5       which globally will be [norV]
6       although they [norO]
7       with minimums [minV]
8       although they [minO]
9       and maximums [maxV]
10      , despite they [maxO]
11  .
```

We have broken the template into multiple lines to improve clarity. The text segments given in lines 2-10 are optional template segments. Please note that lines 3-4 capture a single optional segment that has been broken into multiple lines due to its length. Lines 1 and 11 are fixed template segments. These two segments are both captured via the *static* type even though they differ linguistically (i.e., line 11 is punctuation while line 1 is a phrase).

For each template, we created candidate templates such that they have at most one form and their fragments also have at most one form. Specifically, we translated the candidate English templates and their respective slot fillers into isiXhosa and declared them using ToCT. Using the above-mentioned temperature template for demonstration, we provide two of the twelve translated isiXhosa templates:

```
1  The temperatures will be [minT] for the minimums and [maxT] or the
2  maximums compared to the expected for this time of the year, which
3  globally will be [norV].
4
5  Iqondo eliphantsi lemozulu [minT] kwaye neqondo eliphezulu [maxT]
6  xa lithelekiswa netempritsha elindelekileyo kwelixesha enyakeni,
7  kodwa ndawo yonke itemprisha [norV]
8
9  The temperatures will be [minT] for the minimums and [maxT] for the
10 maximums compared to the expected for this time of the year, although
11 they [norO].
12
13 Iqondo eliphantsi lemozulu [minT] kwaye neqondo eliphezulu [maxT] xa
14 lithelekiswa netempritsha elindelekileyo kwelixesha enyakeni, kodwa
15 ndawo yonke itemprisha [norO].
```

We use the same notation as the GALiWeather templates for the isiXhosa templates, instead of providing the ToCT specification since the above templates do not have polymorphic words. The translation of the English template (lines 1-3) is given in lines 5-7. Similarly, the translation of the second English template (lines 9-11) is given in lines 13-15. In both language's templates, the slots are captured via labels enclosed using braces (e.g., $minT$ in line 1 and $norV$ in line 7). Both isiXhosa templates use

unimorphic words (e.g., *Iqondo* in line 5), phrases (e.g., *eliphantsi lemozulu* also in line 5), slot (e.g., labelled *minT* in line 7), and punctuation (e.g., comma at the end of line 6) but no polymorphic words. While the provided templates do not have polymorphic words, other isiXhosa GALiWeather templates do. More precisely, the number of slots and polymorphic words in each of the isiXhosa templates is shown in Table 6.2. In the table, we use identifiers that were assigned to the templates. The identifiers and names can be found in the templates downloadable from `https://github.com/AdeebNqo/ToCT` (We have also provided the correspondence between the template name and identifier in Table B.1). There is no template assigned to id 8 as that number is assigned to the canned text.

Table 6.2: List of the slots and the number of polymorphic words in each template and the types of slot fillers used. * = Phrase has a polymorphic word. (Abbreviations: Phrase = Phr., PolymorphicWord = Poly., UnimorphicWord = Uni., Subject concord = SC, and Negated subject concord = NSC.)

Templ. id	**Slot(s)**	**# Poly. frag-ments**	**Slot filler types**	**Slot filler example**
1	period1Slot, coverage2Slot, period3Slot, coverage3Slot, coverage1Slot, period2Slot	2	Phr., Uni.	esiphakathini 'towards the middle'
2	coverageSlot	0	Phr., Uni.	elithe gqabagqaba ngamafu 'foggy'
3	period5Slot, coverage5Slot, period4Slot, coverage4Slot, period6Slot	2	Phr., Uni.	ekuqaleni 'at the beginning'
4	coverage5Slot, period4Slot, period7Slot, coverage4Slot, period6Slot	2	Phr., Uni.	elisibakeleyo 'covered'
5	coverage2Slot, coverage1Slot	0	Phr., Uni.	elimathumb' antaka 'cloudy'
6	coverage3Slot, coverage2Slot, coverage1Slot	1	Phr., Uni.	elihle 'clear'
7	coverage5Slot, coverage4Slot, coverage6Slot	0	Phr., Uni.	elisibakeleyo 'covered'
9	maximumTempSlot, variationSlot, minimumTempSlot	0	Poly., Phr.*	[SC]phantsi kakhulu 'very low'

Table 6.2: List of the slots and the number of polymorphic words in each template and the types of slot fillers used. * = Phrase has a polymorphic word. (Abbreviations: Phrase = Phr., PolymorphicWord = Poly., UnimorphicWord = Uni., Subject concord = SC, and Negated subject concord = NSC.)

Templ. id	**Slot(s)**	**# Poly. fragments**	**Slot filler types**	**Slot filler example**
10	maximumTempSlot, oscilationSlot, minimumTempSlot	0	Poly., Phr.*	[SC]qhelekile 'normal'
11	maximumTempSlot, minimumTempSlot, minimumVariationSlot	0	Poly., Phr.*	[NSC]natshintsho 'without changes'
12	maximumTempSlot, minimumTempSlot, minOscilationSlot	0	Poly., Phr.*	[SC]qhelekile 'normal'
13	maximumTempSlot, maxVariationSlot, minimumTempSlot	0	Poly., Phr.*	[SC]iphezulu 'high'
14	maximumTempSlot, maxOscilationSlot, minimumTempSlot	0	Poly., Phr.*	[SC]iphantsi 'low'
15	tempSlot, norVariationSlot	0	Poly., Phr.	[NSC]natshintsho 'no change'
16	tempSlot, oscilationSlot	0	Poly., Phr.	[SC]iphantsi 'low'
17	tempSlot, minimumVariationSlot	0	Poly., Phr.	[SC]hla ngamandla 'in notable decrease'
18	tempSlot, minOscilationSlot	0	Poly., Phr.	[SC]iphezulu 'high'
19	tempSlot, maxVariationSlot	0	Poly., Phr.	[SC]nyuka kancinci 'in slight increase'
20	tempSlot, maxOscilationSlot	0	Poly., Phr.	[SC]phezulu kakhulu 'very high'
21	windSlot, timeSlot	1	Phr., Uni.	amandla ukusuka entshona 'strong from the West'

Table 6.2: List of the slots and the number of polymorphic words in each template and the types of slot fillers used. * = Phrase has a polymorphic word. (Abbreviations: Phrase = Phr., PolymorphicWord = Poly., UnimorphicWord = Uni., Subject concord = SC, and Negated subject concord = NSC.)

Templ. id	**Slot(s)**	**# Poly. frag-ments**	**Slot filler types**	**Slot filler example**
22	time2Slot, windSlot, time1Slot	1	Uni., Phr.	emini 'in the afternoon'
23	wind3Slot, wind4Slot, time5Slot, time4Slot	2	Phr., Uni.	ngolwesine 'on Thursday'

We include a demonstration of a template that includes a polymorphic word here:

```
English template: There will be [cl1] skies  [p1], [cl2] [p2] and [cl3]
[p3] of this term
IsiXhosa template: Kuzakubakho izulu [cl1] [p1], [cl2] [p2] kunye /cl3/
[p3] /xesha/
```

In the above snippet, we use the notation introduced in the previous snippets. However, since there are polymorphic words, we also introduce new notation in the form of items enclosed with slashes to denote such words. Specifically, in the isiXhosa templates, the word corresponding to coverage (labeled "cl3") and the word for clarifying which period of the term is referred to (labeled "xesha") are captured via, or as part, of a polymorphic word. The final value of the first polymorphic word, `/cl3/`, depends on the value inserted into the slot since the phonological conditioning rules may produce different kinds of values. All the slot fillers translated from the GALiWeather dataset that can be inserted into the polymorphic word's slot start with "eli-", e.g., *elithe gqabagqaba ngamafu* or *elisibakeleyo*. Hence, the application of the phonological conditioning rules produces the same kind changes to the connecting value found in the polymorphic word's affixes. Technically, it is possible that other slot fillers that do not lead with "eli-" could be used, even if unlikely, and in such cases, the phonological conditioning rules may produce different connecting values. The polymorphic word, `/cl3/`, is specified using ToCT in the following manner:

```
<coverage3> a toct:PolymorphicWord
    ; co:index "9"^^xsd:positiveInteger;
    ; toct:hasPart <na>, <coverage3Slot>
    ; toct:hasFirstPart <na>
    ; toct:hasLastPart <coverage3Slot>
    ; toct:hasNextPart <period3Slot> .

```

```
<na> a toct:UnimorphicAffix
    ; toct:hasValue "na"^^xsd:string
    ; co:index "1"^^xsd:positiveInteger
    ; toct:hasNextPart <coverage3Slot> .

<coverage3Slot> a toct:Slot
    ; toct:hasLabel "cl3"^^xsd:string
    ; co:index "2"^^xsd:positiveInteger .
```

Similarly, the value of the `/xesha/` polymorphic word is governed by the noun inserted into the slot denoted `[p3]`. For instance, when the locative nouns 'ekuqaleni$_{15}$' and 'esiphakathini$_{7}$' are each inserted into the slot, then the final word of `xesha` could take the forms *kwemini* and *emini*, respectively. Using the ToCT, `/cl3/` is specified in the following manner:

```
<xesha> a toct:PolymorphicWord
    ; toct:reliesOn <period3Slot>
     ; toct:hasFirstPart <possXesha>
     ; toct:hasLastPart <ministem> .

<possXesha> a toct:Concord
     ; toct:hasLabel "possC"^^xsd:string
     ; co:index "1"^^xsd:positiveInteger
     ; cam:hasConcordType <possConType>
     ; toct:hasNextPart <ministem> .

<ministem> a toct:Root
     ; toct:hasValue "imini"^^xsd:string
     ; co:index "2"^^xsd:positiveInteger .
```

In the above snippets, the prefixes *toct*, *co*, *cam* correspond to ToCT, the Collections Ontology (CO), and Concord Annotation ontology, respectively.

The translation resulted in 22 isiXhosa templates and 1 canned text sentence. The ToCT-captured templates can be downloaded from `https://github.com/AdeebNqo/ToCT`. Using the complete list of templates, we generated 22 sentences by randomly selecting slot fillers and ensuring they are inserted in the semantically correct order. The complete list of texts and their ids are provided in Appendix B.

The 22 generated sentences and the single translated canned text sentence are packaged into a survey available with English or isiXhosa instructions. In the survey, respondents were asked to report their isiXhosa proficiency and asked to rate the fluency and grammaticality of the provided texts on a 5-point scale. By fluency, we refer to the evaluation of the text's goodness through subjective human judgements that are absolute à la [293], as classified by [24]. By grammatically, we refer to "adherence to rules of syntax [...] and overall legitimacy as an [isiXhosa] sentence" [106],

an evaluation of the text's correctness through subjective human judgements that are absolute [24].

In the survey, a single attention check question was included and assigned the identifier T3. The purpose of the attention check is to improve the quality of the responses by identifying participants who are not diligent [215]. The English (lines 1-3) and isiXhosa (lines 5-7) texts used as the checks are as follows:

```
1 This is not a question to be evaluated. It is a question to test whether
2 you are paying attention. For this question only, please select
3 "Unclear" for fluency and "Grammatical" for grammaticality.
4
5 Lo ayingombuzo ekufuneka uvavanywe. Ngumbuzo wokubona ukuba uyayifunda
6 na imibhalo. Kulombuzo qha, sicela ukhethe u "Andinamkhethe" kubuciko
7 kwaye ukhethe u "Igrama irongo kakhulu" kukuthobela igrama.
```

Participants were recruited via a post on the author's account on Twitter[2], the social media website. Only isiXhosa speakers were invited to participate, and the call encouraged participants to recruit other respondents, as a way of snowball sampling [97]. By snowball sampling, we refer to the procedure of sampling individuals from some population and also asking them to extend an invitation to other individuals from the same population to participate. The resulting judgements were analysed using the median and inter-quantile ranges.

The English and isiXhosa instructions that were provided to participants during the surveys are also listed in Appendix B.

6.3.2 Results

There were 18 total responses, of which 16 respondents took the survey with English instructions and 2 with isiXhosa instructions. All respondents were self-reported as L1 isiXhosa speakers. Both the respondents who chose isiXhosa instructions passed the attention check and of the 16 respondents who chose English instructions, 2 failed the attention check. Henceforth, the results will only be presented for the 16 who passed the attention check and the attention check (T3) will not be used. As can be observed in Figure 6.3, there are a few texts judged as not being fluent by the respondents. The fluency and grammaticality judgements are presented in Figure 6.3 and Figure 6.4 respectively.

Out of the 23 judged texts, 13 were perceived as fluent and grammatically correct while there is no consensus on the rest. The reasons for the lack of a consensus

[2] https://twitter.com/

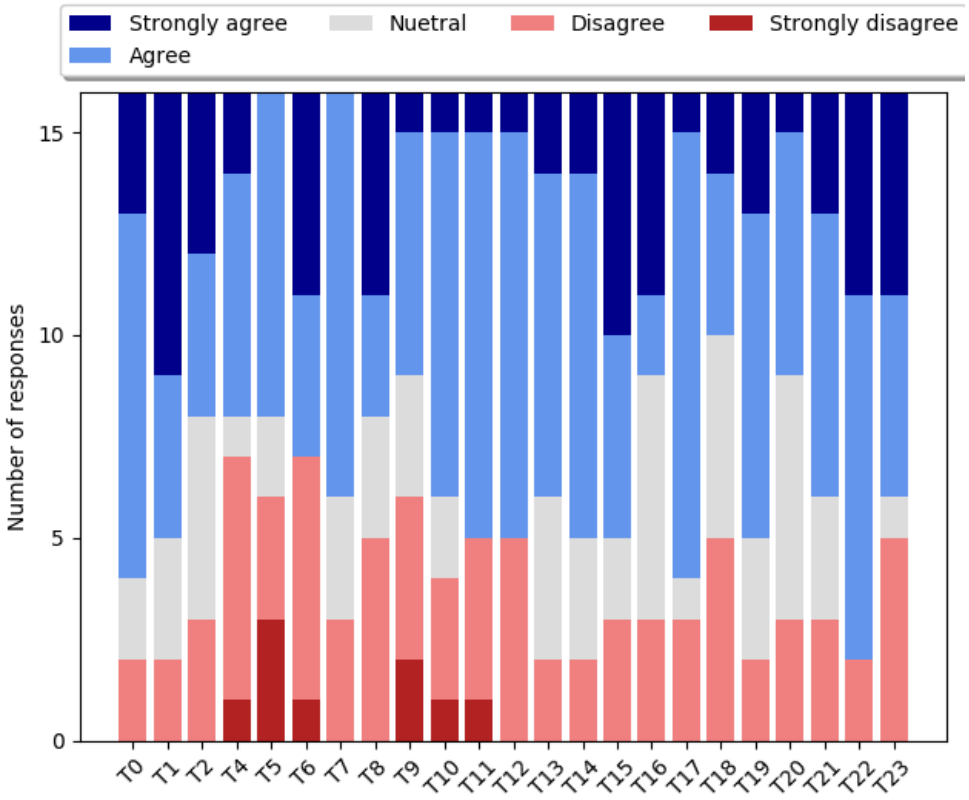


Figure 6.3: Fluency judgements for each of the 23 weather forecast texts

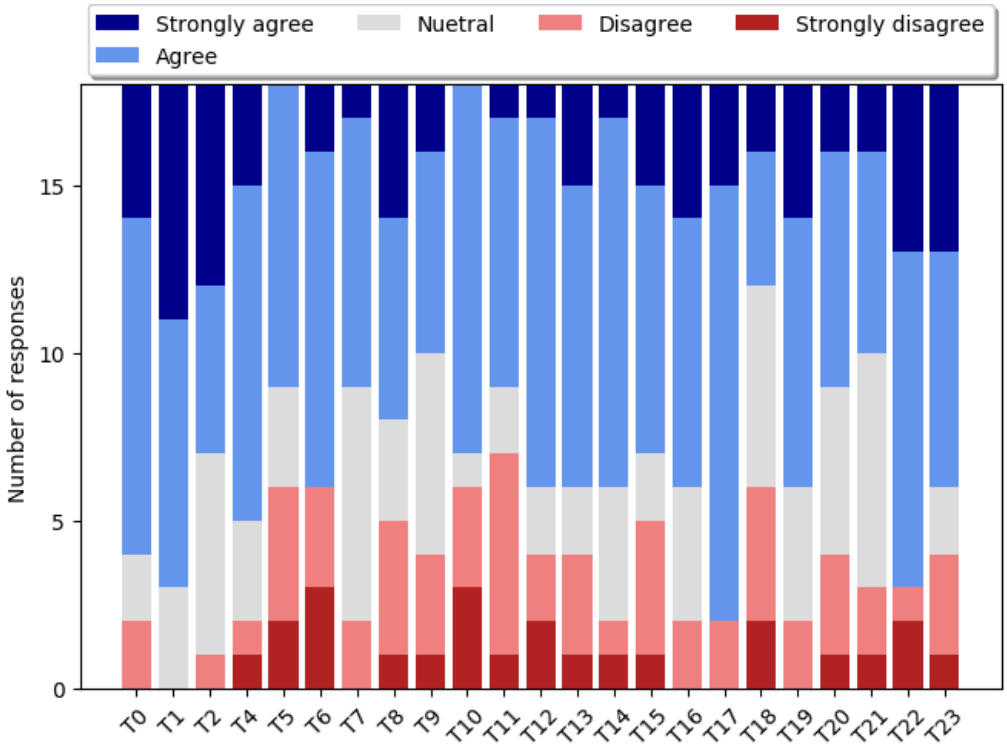


Figure 6.4: Grammaticality judgements for each of the 23 weather forecast texts

are likely to be difficulties judging some texts without surrounding text that provide context and differences in dialects. These issues will be discussed in detail in Section 6.5.

The median (fluent) and inter-quantile ranges (between 0.25 - 2) show that respondents judged 13 sentences as fluent (T0, T1, T7, T10, T11, T12, T13, T14, T15, T17, T19, T21, and T22). The aforementioned texts are as follows:

```
T0: Kuzakubakho izulu elisibakeleyo ekupheleni, elithe gqabagqaba
     ngamafu esiphakathini kunye nelimathumb' antaka ekupheleni kwemini
T1: Kuzakubakho izulu elisibakeleyo jikelele imini yonke
T7: Kuzakubakho izulu elihle kubekho namaxesha apho kukho izulu
     elimathumb' antaka okanye elisibakeleyo
T10: Iqondo eliphantsi lemozulu liphantsi kakhulu kwaye neqondo
     eliphezulu liphantsi kakhulu xa lithelekiswa netempritsha
     elindelekileyo kwelixesha
     enyakeni, kodwa ndawo yonke itemprisha izakunyuka isehla
T11: Iqondo eliphantsi lemozulu liphantsi kakhulu kwaye iqondo
     eliphezulu liphantsi kakhulu xa lithelekiswa nethempritsha
     elindelekileyo kwelixesha enyakeni, iqondo eliphantsi lona
     linyuka ngamandla ekuhambeni kwexesha
T12: Iqondo eliphantsi lemozulu liphezulu kakhulu kwaye iqondo
     eliphezulu liphantsi kakhulu xa lithelekiswa nethempritsha
     elindelekileyo kwelixesha enyakeni, nangona amaqondo
     azakunyuka esehla ekuhambeni kwexesha
T13: Iqondo eliphantsi lemozulu liphantsi kakhulu kwaye iqondo
     eliphezulu liphezulu kakhulu xa lithelekiswa nethempritsha
     elindelekileyo kwelixesha enyakeni, kwaye iqondo eliphezulu
     linyuka kancinci-kancinci ekuhambeni kwexesha
T14: Iqondo eliphantsi lemozulu liphezulu kakhulu kwaye iqondo
     eliphezulu liphantsi kakhulu xa lithelekiswa nethempritsha
     elindelekileyo kwelixesha enyakeni, nangona amaqondo azakunyuka
     esehla
T15: Amaqondo obushushu azakuba ephantsi kakhulu ngelixesha enyakeni,
     kodwa ndawo yonke itemprisha inyuka ngamandla
T17: Amaqondo obushushu azakuba ephezulu kakhulu ngelixesha enyakeni,
     iqondo eliphantsi lona lihla kancinci
T19: Amaqondo obushushu azakuba ephantsi kakhulu ngelixesha enyakeni,
     kwaye amaqondo aphezulu anyuka kancinci ekuhambeni kwexesha
T21: Silindele umoya onamandla amakhulu ukusuka emantla ngomvulo
T22: Silindele imimoya ezakuba namandla amakhulu ukusuka emzantsi-mpuma
     ngomvulo
```

Likewise, the median (grammatically correct) and inter-quantile ranges (between 0 - 2) show that respondents judged 13 texts as being grammatically correct (T0, T1, T4, T6, T12, T13, T14, T15, T16, T17, T19, T22, and T23). The texts that are judged as being both fluent and grammatically correct have variable lengths, they are direct, and have no vague fragments. This can be seen in the following texts, repeated here with English translations:

```
T14: Iqondo eliphantsi lemozulu liphezulu kakhulu kwaye iqondo
eliphezulu liphantsi kakhulu xa lithelekiswa nethemprisha
elindelekileyo kwelixesha enyakeni, nangona amaqondo azakunyuka
esehla
English trans: The temperatures will be very high for the minimums and
very low for the maximums compared to the expected for this time
of the year, even though they will oscillate

T22: Silindele imimoya ezakuba namandla amakhulu ukusuka
emzantsi-mpuma ngomvulo
English trans: We expect winds which will be strong
from the South-East on Monday
```

The first text (T14) is generated by a template that does not have a polymorphic word. However, its first slot is inserted with a polymorphic word whose final value is *liphezulu* 'high' since it is in agreement with *Iqondo* 'temperature'. The relevant fragment of the template that generated T4 is illustrated in the following listing:

```
<temp6Template> a toct:Template
    ; toct:supportsLanguage <cpt_xhosa>
    ; toct:hasPart <qondo1>, <phantsi>,
     <minimumTempSlot>, <kwaye>,  <qondo2>, <phezulu>,
     <maximumTempSlot>, <thelekiswa>, <comma>, <ngona>,
     <qondo3>, <maxOscilationSlot>
    ; toct:hasFirstPart <qondo1>
    ; toct:hasLastPart <maxOscilationSlot> .

<cpt_xhosa> a mola:Dialect
    ; mola:isFamily <isiXhosa> .

<qondo1> a toct:UnimorphicWord
    ; toct:hasValue "Iqondo"^^xsd:string
    ; co:index "1"^^xsd:positiveInteger
    ; toct:hasNextPart <phantsi>
    ; toct:controls <i>
    ; ncs:hasNounClass <nounClass5Type> .

<phantsi> a toct:Phrase
    ; toct:hasValue "eliphantsi lemozulu"^^xsd:string
    ; co:index "2"^^xsd:positiveInteger
    ; toct:hasNextPart <minimumTempSlot> .
```

The second example text (T22) is generated by a template with 3 fragments: a phrase with three unimorphic words (i.e., *Silindele imimoya ezakuba*), a polymorphic word that requires phonological conditioning to form its final form (i.e., *na*⟨*wind+direction*⟩), and a slot (i.e., ⟨*day*⟩).

When we define Disagree = {1,2}, Undecided = {3}, and Agree = {4,5} for both the collected fluency and grammaticality judgements, then we see that there is no evidence that the other sentences are perceived as being unclear or grammaticality incorrect,

due to absence of consensus among the survey participants. When analysing the results, we see that a Student's t-test shows that there is no significant difference in the number of negative and neutral judgements vs. positive judgements for the texts for both fluency ($\alpha = 0.05$, $p = 1$) and grammaticality ($\alpha = 0.05$, $p = 0.509$).

6.3.3 Summary

Table 6.2 shows that a number of templates (32%) have polymorphic fragments; therefore, they are outside the competence of English-oriented templates. While the rest of the templates (15/22) do not have a polymorphic fragment, Table 6.2 shows that of those 15, only three (template identifiers 2, 5, and 7) do not use polymorphic words as slot fillers. Such slot fillers are not possible in simple templates. The use of polymorphic words, as slot fillers, can be seen with the words *azakunyuka* and *awazokunyuka*, as they are produced from the same slot filler since they have the same lemma *-nyuk-* 'rise'. Using polymorphic words for slot fillers stands in contrast with listing all surface form variants, as done in lexicographic models such as Bantu Language Model (BLM) [31] or English-oriented template models. Instead, the final surface form of the word is determined by the surface realiser.

6.4 OWL Simplified isiZulu

Having built a data-to-text generation system for isiXhosa, we now turn to build a knowledge-to-text generation system for isiZulu using ToCT and the rest of the modules. Specifically, we demonstrate that the created architecture and its actualised components generate good quality questions from an ontology.

There are no existing isiZulu Controlled Natural Languages (CNLs) that can be for question generation from an ontology. As such, we designed a CNL to be used by the isiZulu verbaliser in an incremental and bottom-up fashion. We began by selecting the first set of OWL constructors to support by re-purposing an existing English CNL, rewriting its templates into yes/no question templates, and translate them into isiZulu. The chosen CNL is Web Ontology Language (OWL) Simplified English [225]. The OWLSIZ templates we have created from them are provided in Table 6.3. In Table, we use a box around a sequence of affixes to illustrate a decomposed word whose underlying morphemes are provided sequentially. We also use an arrow from a concord to the noun that controls its value. In template 10, for instance, the third word is made up of two affixes: the subject concord and *-odwa*. The subject concord's

value depends on the noun class of the value inserted into the class slot (i.e., {C}) and there are 17 such noun classes in isiZulu. When the subject concord's value is inserted then phonological conditioning may be applied when combining the two affixes since isiZulu does not permit consecutive vowels. For instance, when noun class 15's *ku* is appended to *odwa* we obtain *kodwa*.

Table 6.3: List of English question templates and OWL Simplified IsiZulu templates. Abbreviations: I = individual, C = class, DP = data property, SC = subject concord, COP = copula prefix, and RelC = relative concord (Adapted and translated from OWL Simplified English [225])

-	Axiom	IsiZulu template and English approximation
1.	`SubClassOf(C1 C2)`	Ingabe {SC} onke {C1} {SC} {COP} {C2}? Is every {C1} a(n) {C2}?
1.1	`SubClassOf(C1 CE)`	Ingabe {SC} onke {C1} {SC} {CE}? Does/Is every {C1} {CE}?
2	`ClassAssertion(C I)`	Ingabe {I} {SC} {COP} {C}? Is {I} a(n) {C}?
3	`ObjectProperty Assertion(OP I1 I2)`	{I1} {SC} {OP} {I2} ? {I1} {OP} {I2}?
4	`Equivalent Classes(C1 C2)`	Ingabe noma yi {OC} phi {C2} {SC} {COP} {C1}? Is every {C2} a(n) {C1}?
5	`Disjoint Classes(C1 C2)`	A {SC} kho yini {C1} {RelC} {Cop} {C2}? Is there no {C1} that is a(n) {C2}?
6	`ObjectSome ValuesFrom(OP C)`	{OP} {C}? {OP} a(n) {C}?
6.1	`ObjectSome ValuesFrom(OP CE)`	{OP} {CE}? {OP} a {CE}?
6.2	`ObjectSomeValues From(`OP_{noun} `C)`	yi {OP} {LocPre} {C} ini? {OP_{noun}}s a(n) {C}?

Table 6.3: List of English question templates and OWL Simplified IsiZulu templates. Abbreviations: I = individual, C = class, DP = data property, SC = subject concord, COP = copula prefix, and RelC = relative concord (Adapted and translated from OWL Simplified English [225])

-	Axiom	IsiZulu template and English approximation
6.3	`ObjectSomeValuesFrom`	{OP} {LocPre} {C} ini?
		-
6.4	`ObjectSomeValuesFrom(OP`$_{noun}$ `C)`	yi {OP}{LocPre} {C}?
		{OP$_{noun}$}s a(n) {C}?
7	`ObjectHasValue(OP I)`	{OP} {I}?
		{OP} {I}?
8	`DataPropertyAssertion(DP I L)`	Ingabe {I} {DP} {L}?
		Does/Will/Did/Is {I} {DP} {L}?
9	`DataHasValue(DP L)`	{DP} {L}?
		{DP} {L}?
10	`ObjectAllValuesFrom(OP C)`	{OP} {C}{SC} odwa
		{OP} only {C}?
11	`ObjectExactCardinality(N OP C)`	{OP} {C}{RelC} yi- {N} ncamashi/ngqo ?
		{OP} exactly {N} {C}?
12	`ObjectMinCardinality(N OP C)`	{OP} {C}{RelC} ngaphezulu kuka-{N} ?
		{OP} at least {N} {C}?
13	`ObjectMaxCardinality(N OP C)`	{OP} {C}{RelC} mbalwa ku-{N} ?
		{OP} at most {N} {C}?

Some of the constructors shown in Table 6.3 have different templates used in certain circumstances. Specifically, Template 1 verbalises `SubClassOf` axioms where the range is a named class. Template 1.1 handles cases where the range is a class expression. There are four different templates for the `ObjectSomeValuesFrom` class expressions. Template 6.4 is applicable where the object property is a noun and the OWL class belongs to noun classes 1, 1a, 2, and 2a. Template 6.3 is applicable when the object property is the containment part-whole relation [134] and template 6.2 is used when the object property is a noun. Template 6 and 6.1 are applicable when the range is an OWL class and class expression respectively. The verbaliser has a rule that chooses

an appropriate template when given a class expression.

The created isiZulu templates use concords (consult Section 2.1 for a detailed categorisation of concords), copulas and locative prefixes. These are polymorphic affixes whose values change depending on the noun in which they are found in the case of the locative prefix and copula or due another noun in the sentence in the case of concords. To illustrate use of the isiZulu templates, then let us consider how to verbalise an axiom of the type `A` $\sqsubseteq$ `B`, e.g., `ihebhu` $\sqsubseteq$ `umuthi` and in OWL functional syntax style, `SubClassOf(ihebhu umuthi)`. Template 1 would be chosen, since the `B` in the position of C2 is a named class. One would then obtain the following text *Ingabe lonke ihebhu lingumuthi?* where the concords and copula are underlined. The concords are the *lo-* (from *li-* + *-o-*) and *li-* components of the highlighted text, which are governed by the noun class of the first noun, *ihebhu* 'herb', and the copula is the *-ng-* component, which is determined by the first character of the second noun, *u-*.

6.4.1 Verbaliser implementation

We designed the OWLSIZ verbaliser software to abide by the architecture diagram given in the architecture diagram shown in Figure 6.5. The initial implementation used the architectured described in [169]. Both implementations use Java and the OWL API[3] library to parse the ontology. We will discuss the structure selection and realisation components in the remainder of this section.

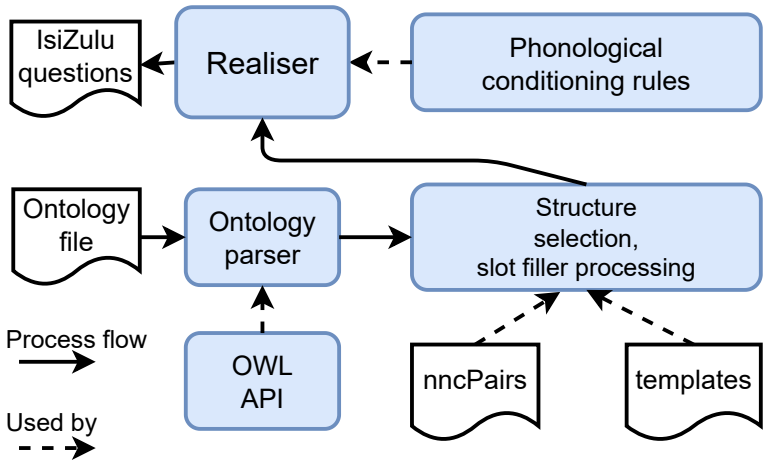


Figure 6.5: Software architecture used by the OWLSIZ verbaliser (Abbreviations: Processing = Proc. and Phonological conditioning = Phon. cond.)

[3] `https://github.com/owlcs/owlapi`

The structure selection module has a few rules for deciding which template to use for each supported axiom type. Out of all the supported OWL constructs, only one logical axiom (`SubClassOf`) and one class expression (`ObjectSomeValuesFrom`) have multiple template forms. A single rule is used by the verbaliser to choose between the multiple forms of templates for each constructor type. The module also does some pre-processing of the slot fillers extracted from the input. This entails basic cleaning, such as removing underscores and replacing them with spaces, and noun class resolution for the nouns. It also uses nncPairs, the file *nncPairs.txt*, for determining whether an object property is or contains a noun and to retrieve the noun class of the noun, which is reused from Keet et al.'s [135] isiZulu verbaliser.

The realisation engine used by the realiser also implements a noun class identifier. The current implementation of the module also uses the *nncPairs.txt*[4] from Keet et al.'s [135]. We extended the file by adding 8 words (we use subscripts to denote each word's noun class) : *umakhalekhukhwini* 'cellphone', *uZola* 'Zola', *iNokia 3310* 'Nokia 3310', *ifoni* 'phone', *umfundi* 'learner/student', *ukubhukuda* 'swimming', *ihebhu* 'herb', and *umdlalo* 'game'. We also changed the noun class annotations of 6 nouns: *ufulawa*$_{3a}$ 'flour', *amanzi*$_{6}$ 'water', *irhaba*$_{9}$ 'Common sowthistle', *ivazi*$_{9}$ 'vase', *ithiyetha yokuhlinzela*$_{9}$ 'operating theather', and *indoda*$_{5}$ 'man' as we disagreed with their original classification.

The surface realiser takes a template and slot fillers, captured via ToCT, and inserts the fillers into their respective slots, resolves the values of concords, locatives, and copulas in polymorphic words, and then forms words by appending the affixes together while relying on isiZulu phonological conditioning rules taken from [204, 252, 227, 275, 222].

6.4.2 Evaluation procedure

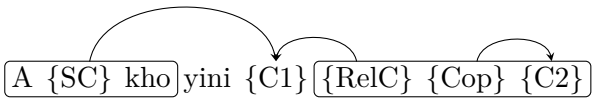
In order to evaluate the quality of questions generated by the verbaliser, we make use of the test ontology presented in [135]. We extended the ontology's 82 logical axioms with 12 axioms to ensure coverage for OWLSIZ: `ClassAssertion` (2), `ObjectProperty` (2), `ObjectPropertyAssertion` (1), `EquivalentClasses` (1), `Data Property` (1), `ObjectAllValuesFrom` (1), `ObjectExactCardinality` (1), `Object MinCardinality` (1), `DataPropertyAssertion` (1), and `ObjectMaxCardinality` (1). This

[4] `https://raw.githubusercontent.com/mkeet/GENIproject/master/isiZuluVerbaliser/OntologyVerbaliser_Zu/nncPairs.txt`

update resulted in an ontology with 91 axioms (it is not 94 because 3 axioms were used to constrain the existing axioms).

For internal evaluation, we verbalised the ontology and categorized the input statements into *verbalisable* and *unverbalisable*. These two classes capture our verbaliser's supported and unsupported axioms. All the unsupported axioms were excluded from the final evaluation. Of those excluded axioms, there were 15 statements produced by them, and there were two main causes why they were categorised as *unverbalisable*. Ten statements had nouns whose noun classes were unresolved by verbaliser's noun class identifier. The problematic nouns included "isampula igazi", obtained from the test ontology's `isampula_igazi`, which ought to have been "isampula yegazi" ('blood sample') in noun class 5. Second, the other five statements included unsupported axiom types (e.g., `ObjectComplementOf`). This is a consequence of using OWL Simplified English as a guide for limiting the scope. The complete list of unverbalisable input is given in Table 6.4.

For each of the *verbalisable* axioms, we analysed their corresponding text to determine whether there are phonological conditioning errors, morphological agreement errors, and any other grammatical error. The surface realiser's error detection module did not find any logical inconsistencies. Out of the total 91 axioms, 76 were verbalisable axioms and their corresponding texts, 75 texts are free of morphological agreement errors, phonological conditioning errors, or any other grammatical errors. For instance, template 5 is selected when verbalising DisjointClasses(isidlanyama isidlazitshalo) and that can be seen in the following snippet:

Input :	*DisjointClasses*(*isidlanyama isidlazitshalo*)
Template :	
IsiZulu output :	*asikho yini isidlanyama esiyisidlazitshalo*?
English approx. output :	*Is there no carnivore that is a herbivore*?

In the isiZulu output, *isidlazitshalo* is prefixed with *esi-* and *-y-* where *esi-* is a relative concord that would have a different value if *isidlanyama* was not used (e.g, it would be *eli-* if a noun that belong to class 5 was used instead). Similarly, *-y-* is the copulative prefix value and if the value inserted into {C2} had *u*, *o*, or *a* for a preceding vowel then *ng-* would be used instead.

For external evaluation, we verbalised the ontology and packaged the resulting questions into a survey. Participants were recruited via snowball sampling [97] using

Table 6.4: List of OWL input statements that are not verbalisable by the system and the reasons of the inability

Input	**Reason**
SubClassOf (<testonto#isampula_igazi> ObjectSomeValuesFrom (<testonto#isiqephu> <testonto#igazi>))	Noun class of "isampula igazi" cannot be resolved
SubClassOf (<testonto#ucezu_isinkwa> ObjectSomeValuesFrom (<testonto#isiqephu> <testonto#isinkwa>))	Noun class of "ucezu isinkwa" cannot be resolved
SubClassOf (<testonto#ukhilimu_oyiqhwa> ObjectSomeValuesFrom (<testonto#eeee> <testonto #ukukhota_ukhilimu_ oyiqhwa>))	Noun class of "ukukhota ukhilimu oyiqhwa" cannot be resolved
SubClassOf (<testonto #ukukhota_-ukhilimu_oyiqhwa> ObjectSomeValuesFrom (<testonto#isiqephu> <testonto#ukhilimu_-oyiqhwa>))	noun class of "ukukhota ukhilimu oyiqhwa" cannot be resolved
SubClassOf (<testonto#okubomvu> <testonto#umbala>)	noun class of "umbala" cannot be resolved
SubClassOf (<testonto#isifundo_sezibalo> <testonto#isifundo>)	noun class of "isifundo sezibalo" cannot be resolved
SubClassOf (<testonto#igazi> ObjectSomeValuesFrom(<testonto#eeee> <testonto#isampula_igazi>))	noun class of "isampula igazi" cannot be resolved
SubClassOf (<testonto#isinkwa> ObjectSomeValuesFrom(<testonto#eeee> <testonto#ucezu_isinkwa>))	noun class of "ucezu isinkwa" cannot be resolved
SubClassOf (<testonto#UMphathi> <testonto#umsebenzi>)	noun class of "UMphathi" cannot be resolved
SubClassOf (<testonto#iphilisi> ObjectComplementOf (ObjectSomeValuesFrom (<testonto#enziwe> <testonto#umshobingo>)))	ObjectComplementOf not in scope
SubClassOf (<testonto#ugogo> ObjectComplementOf (ObjectSomeValuesFrom(<testonto#dla> <testonto#i-apula>)))	ObjectComplementOf not in scope
SubClassOf (<testonto#umuntu> ObjectComplementOf (ObjectSomeValuesFrom(<testonto#feza> <testonto#umsebenzi_onqunyiwe>)))	ObjectComplementOf not in scope
SubClassOf (<testonto#ivazi> ObjectComplementOf(ObjectSomeValuesFrom (<testonto#akhiwe> <testonto#amanzi>)))	ObjectComplementOf not in scope
SubClassOf (<testonto#ingwe> ObjectComplementOf (ObjectSomeValuesFrom(<testonto#dla> <testonto#i-apula>)))	ObjectComplementOf not in scope

the author's Twitter and WhatsApp accounts. Following Keet and Khumalo [136], we were interested in determining the extent to which OWLSIZ "generates grammatically correct sentences" [136]. We do so by getting human judgements of the texts' grammatically and acceptability. By grammatically, we are still referring to "adherence to rules of syntax [...] and overall legitimacy as an [isiZulu] sentence" [106] and by acceptability, we are referring to a goodness criterion used to determine whether the text is acceptable or understandable, even if it may have a few grammatical inconsistencies. Specifically, the participants were asked to judge the quality of each question by choosing either "grammatical and acceptable", "grammatical and ambiguous", "ungrammatical and understandable", or "ungrammatical and unacceptable". To ensure high-quality judgements, we ensured that each participant did not judge more than 40 sentences by randomly dividing the 76 texts into two surveys each containing 38 texts (henceforth, surveys A and B). Survey A includes texts generated by templates 3, 10, 12, and 13 while survey B does not and survey B includes texts generated by templates 4, 8, and 11 while survey A does not. Both surveys include the texts generated by all the other templates. Participants were randomly assigned to a survey.

The *verbalisable* input to the verbaliser, the template chosen for each input, and the resulting question are given the Appendix B. We provide an example of the generated question and its input:

Input :	$SubClassOf(< testonto\#ukubhukuda >< testonto\#umdlalo >)$
Template :	Ingabe {SC} onke {C1} {SC} {COP} {C2} ?
Output :	*Ingabe konke ukubhukuda kungumdlalo?*

In the example above, we use 'testonto' prefixes to refer to Keet and Khumalo's test ontology. The value of the second and last words in the output depend on the noun class of the input. The above template is captured using ToCT in the following manner:

```
<templ1> a toct:Template
    ; toct:supportsLanguage <translZu>
    ; toct:hasFirstPart <ngabe>
    ; toct:hasLastPart  <qmark>
    ; toct:hasPart <onke>, <c1>, <c2> .

<translZu> a mola:Dialect
    ; mola:isFamily <isiZulu> .

<ngabe> a toct:UnimorphicWord
```

```
11      ; toct:hasValue "Ingabe"^^xsd:string
12      ; toct:hasNextPart <onke> .
13
14  <onke> a toct:PolymorphicWord
15      ; toct:reliesOn <c1>
16      ; toct:hasFirstPart <bo>
17      ; toct:hasLastPart  <onkeRoot>
18      ; toct:hasNextPart <c1> .
19
20  <bo> a toct:Concord
21      ; cao:hasConcordType <subjCConType>
22      ; toct:hasLabel "subjC"^^xsd:string
23      ; toct:hasNextPart <onkeRoot> .
24
25  <onkeRoot> a toct:Root
26      ; toct:hasValue "onke"^^xsd:string .
27
28  <c1> a toct:Slot
29      ; toct:hasLabel "C1"^^xsd:string
30      ; toct:hasNextPart <c2> .
31
32  <c2> a toct:PolymorphicWord
33      ; toct:reliesOn <c1>
34      ; toct:hasFirstPart <c2Sub>
35      ; toct:hasLastPart  <c2Slot>
36      ; toct:hasNextPart <qmark> .
37
38  <c2Sub> a toct:Concord
39      ; toct:hasLabel "subjC"^^xsd:string
40      ; cao:hasConcordType <subjCConType>
41      ; toct:hasNextPart <cop> .
42
43  <cop> a toct:Copula
44      ; toct:hasLabel "COP"^^xsd:string
45      ; toct:hasNextPart <c2Slot> .
46
47  <c2Slot> a toct:Slot
48      ; toct:hasLabel "C2"^^xsd:string .
49
50  <qmark> a toct:Punctuation
51      ; toct:hasValue "?"^^xsd:string .
```

The first word is captured a unimorphic word (lines 10-12), since the second word's value depends on the noun inserted into the slot, it is captured via a polymorphic word (lines 14-18), and the last word is also captured via a polymorphic word (lines 32-36). The polymorphic words have slots whose types is captured via the concord annotation ontology (see lines 21 and 40).

We now turn to discuss the results of the evaluation.

Table 6.5: Number of participants' judgements. Abbreviations: Pct. = percent, accept. = acceptable, understand. = understandable, gramm. = grammatical, ambig. = ambiguous, ungramm. = ungrammatical

Survey	**Gramm. + ambig.**	**Gramm. + accept.**	**Ungramm. + understand.**	**Ungramm. + unaccept.**
A	17	41	6	12
B	23	78	19	32
A+B	40	119	25	44
A+B Pct.	18%	52%	11%	19%

6.4.3 Results of external evaluation

Only Template 8 yielded text with grammatical errors. When verbalising `DataPropertyAssertion(neminyaka uZola 50)`, the system generated the following output *Ingabe uZola neminyaka 50?* 'Is Zola aged 50?'. This is lacking agreement markers; hence it is grammatically incorrect. The correct should be *Ingabe uZola [SC]neminyaka [RelC]ngu-50?* where the subject concord (SC) depends on the individual (*uZola*) and the relative concord (RelC) depends on the noun found in the object property (i.e., *iminyaka*). This was corrected in the ToCT captured templates and they can downloaded from `https://github.com/AdeebNqo/ToCT`.

External evaluation received a total of six participants. Of those six, two took part in Survey A (1 L1 isiZulu speaker and 1 L2 speaker) and the four took part in Survey B (all are L1 isiZulu speakers). The two participants from survey A made a total of 76 judgements. Survey B participants made a total of 152 judgements (38 per participant). The separation of the participants' judgements into their respective categories is listed in Table 6.5.

We also calculated the number of texts that all participants judge as being grammatically correct irrespective of whether they are acceptable or ambiguous. This was calculated by counting the number of texts where participants judged them as any combination of 'grammatical and ambiguous' and 'grammatical and acceptable'. Survey A participants agreed with their judgements in 16/38 texts and survey B participants agreed in 18/38 texts. By agree, we mean that they selected the same options among the four for each text. We also investigated whether the disagreement was not as a result of text length since some participants may prefer compact text. We found that the disagreement is not due to differences in text length since the texts for which the participants agree and disagree have texts of similar length (agree =

average of 4 words and disagree = 5 average words), save for a single outlier with 7 words in the texts for which they disagree. The number of sentences for which the participants judge in the affirmative for grammaticality, understandability, and acceptability are given in Table 6.6.

There are 7/38 texts where participants agreed that they are either ambiguous or unacceptable in Survey A and there are 10/38 such texts in Survey B.

6.4.4 Summary

ToCT is able to capture the 14/18 templates that are outside the scope of English-oriented templates. Of the 18 total templates, only four templates (6, 7, 8, 9) only have fixed words and slots only. The rest of the templates either support sub-lexical constituents, word-to-word dependencies, or both. Technically, English oriented template models could be stretched to cater for the templates that only have sub-lexical constituents and no dependencies (6.1, 6.2, 6.3, and 6.4 in Table 6.3). For Nguni languages, this approach forces one to pair the required grammar rules that were introduced in Chapter 2 in an orthogonal fashion. This approach may be sufficient for small application domains; however, in order to produce reusable tools that are easy to maintain then the organisation of the various components must be reached via deliberate steps.

6.5 Discussion

We now turn to discuss ToCT's scope and its ability to capture the various elements found in the isiXhosa GaliWeather and OWLSIZ template collections. We also discuss the grammaticality and acceptability of the generated text.

A majority of OWLSIZ's texts are judged positively. We use the phrase 'judged positively' to refer to aggregate judgements of understandable, acceptable, and grammatically correct texts. In survey A and B, Table 6.6 shows that there are a few texts where the participants made the same judgement. Nonetheless, since there were only

Table 6.6: Number of text for which participants agree regarding grammaticality, understandability, and acceptability

Survey	**Grammaticality**	**Understandability**	**Acceptability**
A	23	0	12
B	18	2	9

7/38 (Survey A) and 10/38 (Survey B) texts where the participants agreed that the text is either ambiguous or unacceptable then this means that for a majority of the texts (31/38 for Survey A and 28/38), participants had a positive view but differed in that one believed text was acceptable while the other viewed it as understandable. The participants also agreed that most of Survey A's texts (61%) are grammatically correct.

A closer analysis of the few texts where there is agreement between participants that the texts are either ambiguous or unacceptable, it is likely that the texts are of good quality but participants misunderstood how to evaluate the text. For instance, when given text that reflects an unacceptable conceptualisation of the real world, then participants may be selecting "ungrammatical and unacceptable" to reflect the unacceptability of the conceptualisation, as opposed to evaluating the quality of the text. This is suspected in the evaluation of the question *Ingabe lonke ibhotela lenza ifoni eliyi-1 ncamashi?* ('Does every butter make exactly 1 phone?') where three participants selected "Ungrammatical and unacceptable" and one selected "Grammatical and ambiguous", even though the text is of good quality in this case, as judged by the present author (an L2 isiZulu speaker). In the context of validating knowledge contained in an ontology, this is a valid question. However, when analysed from a purely linguistic perspective then it is not valid since there is no agreement between the mass noun (i.e., butter) and countable noun (i.e., phone).

This affirms the observation that since natural language is not precise, human participants who have no modelling experience are likely to assume unspecified knowledge and that may lead to errors [238, 226]. It motivates the use of primers to ensure that participants only evaluate text and do not commit to a particular conceptualisation of the world since the ontology's knowledge may capture a different world. Such a difference may be due to erroneous modelling (e.g., the false assertion that a professor is a part of some university in a strict[5] mereological sense), a conceptualisation that is known to experts in a particular domain but goes against common-sense (e.g., dogs salivate as a way of sweating), or any other reason.

Returning to the isiXhosa GaliWeather system, there is agreement among the participants that over 50% of the text generated are fluent and grammatically correct. The rest of the texts (10/23) are not determined to be unclear and ungrammatical, instead there is no consensus regarding their quality. There are numerous possibilities as to why there is no consensus over these texts. For instance, it may be confusing to

[5]cf. Keet and Artale's [132] meronymic part-whole relation

imagine a context in which one would expect T20 from a forecast hence it does not seem fluent and grammatical to some respondents.

```
T20: Amaqondo obushushu azakuba ephantsi kakhulu ngelixesha enyakeni,
nangona amaqondo awazokunyuka esehla ekuhambeni kwexesha.
English trans: The temperatures will be low for this period of the
year, but they will not oscillate over time.
```

The isiXhosa equivalent of the phrase "for this period of the year" may be seen as confusing, without additional context, since does not explicitly state the period being compared. The isiXhosa text also uses 'amaqondo" twice instead of a referring expression on the second time.

The reason for the lack of consensus cannot be morphological agreement and/or phonological conditioning errors because analysis by the author, who is an L1 speaker, shows that there are none. Alternatively, there may be region-based preferences due to one's dialect. For the created templates, we have used what can be termed as the "official" (recognised as such by the South African constitution) or standardised isiXhosa. Nomlomo [210] (as cited in [253]) has previously pointed out that development of the standard language heavily favours the Gaika[6] dialect (numbered S41D in [173, 174]) for historical reasons. Consequently, speakers of dialects such as Hlubi or Bhaca, for instance, may prefer and easily understand the text if it used different terminology and grammatical constructions, especially since there is evidence that isiXhosa-speaking students struggle with understanding standardized isiXhosa [203]. To demonstrate the dialect issue for our templates, consider the difference in the use of **/ths/** instead of **/th/** between the Bhaca dialect and the official isiXhosa. Nomlomo suggests that the Bhaca dialect uses **/ths/** while official isiXhosa uses **/th/**. For instance, the official isiXhosa' word *ndithi* 'I say' takes the form *ndithsi* in Bhaca [210]. Consequently, it is likely that some participants may have expected GaliWeather's texts T0, T4, and T6 to use *elithse* 'that is' instead of *elithe*, for instance.

Solving the dialect issue when building an NLG system is not trivial, as evidenced by the fact that it has also not been solved in the education sector. South African language teachers and the education research community has had debates on how to best teach students and balance their needs against the expectations and wishes of parents [253]. In the case of NLG, one could try and resolve the matter via selecting evaluation participants and binning them by dialect. This approach assumes that we have the ability to generate the same text for the various dialects. However, binning participants is not practical for Nguni languages because we cannot generate different

[6]Alternative names used by other authors are isiNgqika, Ngqika, or Ncqika.

versions of the same text for the various dialects. This is a consequence of not having up-to-date documentation of the differences in dialects for the considered languages. In addition, creating such documentation is outside the scope of this thesis and is left for future linguists. In general, one possible solution to this issue, first proposed by Alexander [6], is the deliberate harmonisation of Nguni languages in order to create "Standard Nguni"; thus doing away with the problem of multiple dialects. To the best of our knowledge, this suggestion has not been acted upon, partly because the Nguni languages do not have an institution similar to the French Academy. Moving forward, additional work is required to investigate whether Nomlomo's [210] catalogue of lexical and grammatical differences between some of the dialects still hold. Even if their catalogue, or a portion of it, still holds for present day dialects then it would still need to be extended to cover more than a subregion of the Eastern Cape province.

We now turn to compare our work to existing approaches; our work offers a separation between the declaration of the templates and their linearisation; hence, it enables re-usability with respect to the linearisation algorithms. Our approach does not require the recreation of the linearisation algorithms for each new application domain. In addition, unlike the *ad hoc* or augmented templates, we offer a principled approach that allows a clear analysis and purposeful approach to combine templates and rules. This means that our approach can be used to decide how best to combine the two assets to allow re-usability and/or scaffolding. Moreover, since our approach relies on ontologies, it allows the use of automated reasoners for template inconsistency checking.

Overall, the outcomes of the isiXhosa and isiZulu evaluation described above suggest that our approach is able to generate good quality text within the described constraints. The systems used for the evaluations generate text by using templates that are captured using ToCT for isiZulu and isiXhosa.

Chapter 7

Conclusion

The purpose of this thesis was to solve the research problem that was first stated in Section 1.5. We restate it here for convenience:

> To the best of our knowledge, there are no systematic and planned methods of associating templates with CGRs. There are also no specifications of interoperable templates, especially ones that have support for morphologically rich languages. In addition, there are no architectures for creating an easy to maintain and reusable template-based surface realisers. As such, there are no Nguni language surface realisation tools that are easy to maintain, can be used for knowledge-to-text and data-to-text systems, and generate understandable and grammatically correct text.

We set out to solve the research problem via a two-step approach:

- **Step 1**: we presented a method for a pairing of simple templates and CGRs in Chapter 3, an architecture for surface realisers that are easy to maintain and prioritise resource reuse in Chapter 5, and an ontology-based specification for simple templates that support morphologically rich languages in Chapter 4.
- **Step 2**: we then created a partial data-to-text system and a complete knowledge-to-text system for isiZulu and isiXhosa, respectively, and evaluated the quality of their generated texts in Chapter 6. The two systems use a surface realiser that expects templates captured using the Task ontology for CNL Templates (ToCT).

We demonstrate how the theoretical contributions and proof-of-concept tools introduced collectively solve the research problem and answer the research questions in

this chapter.

The rest of chapter is structured in the following manner: Section 7.1 presents a summary of the achievements and we have organised them according to the research questions introduced in Chapter 1, Section 7.2 highlights the main contributions of this thesis, and Section 7.3 presents a number avenues that can be pursued as future work.

7.1 Revisiting research questions

The research that we conducted was driven by specific research questions. We now turn to demonstrate how we have answered each one.

1. To answer research question 1 (*What are the characteristics of systematic methods of pairing templates and computational grammar rules to form augmented templates? How do they enhance the selection of models of templates to support grammatically complex languages?*): our results show that a model-based approach allows one to deliberately choose the kinds of pairing relationships that are to be supported. Consequently, efficient use of resources is attained intentionally instead of being arrived at by coincidence. The answer to this question depends on the answers to its sub-questions:

    - To answer research question 1 part 1a (*How to relate templates with CGRs to support template scaffolding and resource reuse?*): we demonstrated that an approach that begins by creating a general model of the relationships for pairing the two artefacts is necessary. We created a conceptual model whose relationships are selected to support template scaffolding and CGRs reuse in this thesis. We then formalised it using set theory in Section 3.2 and demonstrated in Section 3.4 that seven families of grammar-infused templates exist based on the types of relationships supported using the formalised model.

    - To answer research question 1 part 1b (*How do the* ad hoc *models used by Natural Language Generation (NLG) existing systems differ concerning their support for the features mentioned in sub-question 1a?*): existing systems can be compared based on their support for embedding, partial attachment, and compulsory attachment for pairing simple templates and CGRs. They can be classified as belonging to one of the seven families. We

also demonstrated the utility of the families and the categorisation via an isiZulu use-case of choosing an appropriate approach for pairing template and CGRs in the context of building a NLG system for producing questions in the area of conceptual modelling. While one can create a new template type each time when building a system, our approach makes it possible to make informed decisions when reusing an existing template type.

2. To answer research question 2 (*What are the various surface realisation tasks and how can they be organised to produce surface realisers that are reusable and easy to maintain for Nguni languages?*): we identified structure induction, selection, encoding, linearisation, and ranking as the crucial tasks for comparing surface realisers. These features are important for control and maintainability. The decision is based on our knowledge and analysis of surface realisers in other languages. We analysed existing surface realisers as they are found in multiple domains and eras to identify current and past architectural trends in finer detail. We then used the discovered trends regarding the surface realisation tasks and apriori knowledge regarding maintainable software products to create a surface realisation architecture. The answer to this question also depends on the answers to the corresponding sub-questions:

   - To answer question 2 part 2a (*How do surface realisation tasks, as found in existing NLG architecture comparisons, limit the analysis of existing surface realiser architectures at finer granularity concerning their suitability for Nguni languages?*): we showed that the granularity of the existing architecture analysis does not allow an ultra-fine analysis of each module, especially the surface realiser, since it has focused on high-level tasks required by all modules. Specifically, the tasks identified and used by [68, 196] do not make clear what sub-tasks are required by the surface realiser to do 'ordering'. Finer granularity is crucial for improving surface realisers for Nguni languages since the few attempts at building such components have been *ad hoc* and have not made architectural design decisions a centrality.

   - To answer research question 2 2b (*What are the granular tasks, low-level than the tasks identified in part 2a, and how can they organised to achieve easy to maintain and reusable surface realisers for Nguni languages?*): we identified five low-level tasks (i.e., structure selection, encoding, induction, linearisation, and ranking) in Section 5.2 and mapped each of the tasks

to separate modules. We then created an architecture that supports the most features of a maintainable software product, as defined in the BS ISO/IEC 25010:2011 standard.

- To answer the part of research question 2 part 2c (*What are the algorithm(s), template specification(s), and annotation model(s) that are needed for the relevant tasks mentioned in question 2b in order to produce correct text?*): we determined the need and created the a number of resources. We created an algorithm for the linearisation task. We created a task ontology, called ToCT, formalised in Web Ontology Language (OWL), that can be used to capture simple templates that are also appropriate for Nguni languages since Chapter 3's definition of grammar-infused templates allows the separation of the templates from the engine. Concerning the annotation model, we determined the need for a linguistic annotation ontology for capturing the types of concords. We then created a concord annotation ontology, together with language-specific axiomitisations for isiXhosa and isiZulu, following the framework presented in [55].

3. To answer research question 3 (*Do Nguni NLG systems that use a realiser that organises its modules according to the method established in Q2 generate text that is correct (i.e., fluent, acceptable, and/or grammatically correct) in selected data-to-text and knowledge-to-text scenarios?*): we built a data-to-text system for isiXhosa and a knowledge-to-text for isiZulu as instruments to investigate this question. The templates used in both systems are captured using ToCT, our novel task ontology, and they have multiple polymorphic words; hence, they are outside the competence of simple templates and canned text. We demonstrated that most of the text generated by the systems is judged positively by speakers of isiXhosa and isiZulu through surveys.

7.2 Contributions

This thesis' main contributions lie in the field of NLG, in the sub-area of surface realisation for low-resourced languages. At a finer granularity, they are in the areas of template representation to support interoperability, surface realiser architectures that result in software that is easy to maintain, and combination of templates and CGRs, while prioritising resource re-use, to support low-resourced languages. The

complete list of contributions covers theoretical aspects, proof-of-concept software, ontologies, templates, and a corpus.

Theoretical contributions:

- A model-based approach for pairing templates and CGR, to be used to produce what we have termed grammar-infused templates (introduced in Section 3.2). This contribution stems from work done to answer research question 1.
- An architecture for template-based surface realisers that are easy to maintain to be used for generating African languages, especially ones that are low-resourced. The architecture is also usable for non-African languages (introduced in Chapter 5). This contribution stems from the work done to answer research question 2.
- An ontology, called ToCT, for template specification and supporting interoperability between systems that make use of templates (introduced in Chapter 4). This contribution also stems from our work done to answer research question 2.

Proof-of-concept software, introduced in Chapter 6, and it stems from work done to answer research question 3:

- A template-based surface realiser for isiZulu and isiXhosa whose support has been demonstrated in the areas of data-to-text and knowledge-to-text.
- A generic grammar engine interface that is implemented in Java. We rely on the polymorphism principle from Object-Oriented Programming (OOP) to create a generic base that can be instantiated by any Niger-Congo B (NCB) language.
- Instances of grammar engines for isiXhosa and isiZulu that are implemented using Java.
- An isiZulu question generator called OWLSIZ that is implemented in Java.
- A template parser for templates that are capturing using ToCT and serialised in Terse RDF Triple Language (Turtle).

Linguistic annotation ontologies and templates (introduced in Chapters 4 and 6):

- A task ontology to be used for annotating concord types and its axiomatisations for isiZulu and isiXhosa.

- Templates for generating isiZulu questions from 13 main OWL constructs (i.e., `SubClassOf`, `ClassAssertion`, `ObjectPropertyAssertion`, `EquivalentClasses`, `DisjointClasses`, `ObjectSomeValuesFrom`, `ObjectHasValue`, `DataProperty Assertion`, `DataHasValue`, `ObjectAllValuesFrom`, `ObjectExactCardinality`, `ObjectMin Cardinality`, and `ObjectMaxCardinality`). They can be adapted for other purposes within the area of question generation from ontologies.
- Templates for generating weather forecasts in isiXhosa. They were obtained by translating templates from GaliWeather [234]. They can be used a base for a complete NLG system where the engineer would be responsible for building the appropriate document planning (and some micro-planning rules) for their target audience.

The proof-of-concept software, excluding OWLSIZ verbaliser, can be downloaded from `https://github.com/AdeebNqo/NguniTextGeneration`. The linguistic annotation ontologies, templates, OWLSIZ verbaliser, and corpus can be downloaded from `https://github.com/AdeebNqo/ToCT`.

7.3 Further research

In this section, we provide two avenues for additional work that we have identified.

7.3.1 Template creation and management tools

In this work, we have created the templates manually using a simple text editor and sometimes Protégé, the ontology development environment. This approach is not optimal for new users of ToCT, especially one with no expertise regarding the morphology of NCB languages, hence there is a need for tools to support the effective management and visualisation of the resources. Additional work could be done in creating Graphical user interface (GUI)-based tools that have assistive features such as drag-and-drop, auto-complete, and extracting candidate templates from a corpus (e.g., Proskurnia et al.'s [230] work on English emails). The drag-and-drop feature can be understood as serving the same function as Scratch's [176] blocks but for ToCT's concepts. The exact features to be included in such a tool could be determined and prioritised based on which template creation errors are made frequently by new users. Nonetheless, the primary feature of such a tool could be the rendering/visualisation

of ToCT because while ToCT has several benefits from a computational perspective, its templates are long and not easy to interpret to new users.

7.3.2 Automated tools for approach selection

We have created a classification of existing template types and categorisations of surface realiser architectures. Developers of template-based surface realisers could use the Tables presented in Chapters 3 and 5 as part of their work-flows. This also holds for engineers who are building tools for non-NCB languages since the architectures discussed in Chapter 5 are usable for other languages, for instance. Additional work could be done to build automated tools to assist engineers choose a template model or an architecture without necessarily pouring over Chapters 3 and 5; hence, expediting the process. Moreover, tools that instantiate code that abides by the architecture could be built and integrated into popular Integrated development environments (IDEs). This is somewhat analogous to how the OntOlogy Pitfall Scanner! (OOPS!) expedites the process of ontology validation when compared to an engineer's manual use of the catalogue of pitfalls provided by Poveda-Villalón, Gómez-Pérez, and Suárez-Figueroa [223].

7.3.3 Coverage for other Nguni languages

We analysed the broad landscape of systems that rely on a variety of NLG techniques in Chapters 2 and 3 and we found that most of the systems are usable or are potential candidates for the languages in question rely on *ad hoc* template models. In addition, most of the existing techniques were not targeting NLG applications for low-resourced languages or Southern African languages. As a remedy, we have presented novel models and resources to enable the creation of surface realisation techniques that combine templates and grammar rules for purposes of generating text. However, our work has currently focused on two Nguni languages, namely isiXhosa and isiZulu. For future work, we aim to extend the coverage of the created engines and we also aim to extend the languages covered to include isiNdebele, and siSwati (and their respective dialects).

7.3.4 NLG systems with mixed methods, resource reuse, and evaluation

We plan to investigate how to integrate surface realisers that abide by our architecture as part of larger NLG systems whose other modules may be neural-based. Specifically,

we will investigate the extent to which the language-independent modules introduced by Castro Ferreira et al. [51] can be reused.

We also plan to investigate the ease with which NLG engineers are able to discover and adapt existing templates for new applications.

Lastly, we also plan to consult a sociolinguist with whom we can study the differences in isiZulu and isiXhosa dialects, use the differences to alter our templates to generate texts for each dialect, and conduct a larger evaluation for each dialect.

Appendix A

ToCT Competency questions and queries

Competency questions followed their respectively SPARQL Protocol and RDF Query Language (SPARQL) queries, or notes if they are not in scope or not translatable.

```
CQ1. How many fixed phrasal and lexical segments does [template]
 have?

SELECT distinct ?template (count(?unimorph) as ?numUnimorphWords)
(count(?phrase) as ?numFixedPhrases)
((?numUnimorphWords + ?numFixedPhrases) as ?totalFixedParts)
WHERE {
     {
     ?template a toct:Template .
     }
     UNION
     {
     ?template co:item ?unimorph .
     ?unimorph a toct:UnimorphicWord .
     }
     UNION
     {
     ?template co:item ?phrase .
     ?phrase a toct:Phrase .
     filter not exists { ?phrase toct:hasValue "" } .
     }
}
GROUP BY ?template


CQ2. How many words that depend on others does [template] have?

SELECT distinct ?template (count(distinct ?polymorph) as
      ?totalWordsThatDependOnOthers)
```

```
WHERE {
     ?template a toct:Template .
     ?polymorph a toct:PolymorphicWord .
     ?template co:item ?polymorph .
}
GROUP BY ?template

CQ3. Which properties may result in a change of form between
 [word1] and [word2] where there exists a dependency?

SELECT ?word1 ?word2 ?propType
WHERE {
     ?word1 a toct:UnimorphicWord .
     ?word2 a toct:PolymorphicWord .
     ?conc a toct:Concord .
     ?word2 co:item ?conc .
     ?word1 toct:controls ?conc .
     ?word1 toct:labeledWith ?prop .
     ?label a ?propType .
     ?propType rdfs:subClassOf toct:PropertyClass .
}


CQ4. If there is a dependency between [word] and [word], which
     word is the governor?

SELECT ?governor ?word
WHERE {
     ?governor a toct:UnimorphicWord .
     ?word a toct:PolymorphicWord .
     ?conc a toct:Concord .
     ?word co:item ?conc .
     ?governor toct:controls ?conc .
}


CQ5. Does [word] have a constant base form?

SELECT distinct ?wordWithConstantBase
WHERE {
     ?wordWithConstantBase a toct:PolymorphicWord .
     ?root a toct:Root .
     ?wordWithConstantBase co:item ?root .
}

CQ6. Which grammar rule will be activated when forming [word]
     if its dependenton another word?

Not in scope.

CQ7. Can [word] ever be placed in [slot]?

SELECT ?wordThatCanFillSlot ?slot
WHERE {
```

```
     ?wordThatCanFillSlot a toct:Word .
     ?slot a toct:Slot .
     ?wordThatCanFillSlot toct:fillsIn ?slot .
}

CQ8. Can the word ordering in [template] exist without the
 template?

NO SPARQL

CQ9. Can the word portion`s ordering in [word] exist without
 the word?

NO SPARQL

CQ10. Which words use [grammar rule]?

Not in scope

CQ11. Is [template] grammar-infused?

Not in scope.

CQ12. How many slots does [template] have?

SELECT ?template (count(?slot) as ?slotCount)
WHERE {
     ?template a toct:Template .
     ?slot a toct:Slot .
     ?template co:item ?slot .
}
GROUP BY ?template

CQ13. How many fixed segments have more than one word?

SELECT distinct ?template (count(?phrase)
     as ?NumFixedSegmentsMoreThanOneWord)
WHERE {
     ?template a toct:Template .
     ?phrase toct:memberOf ?template .
}
GROUP BY ?template

CQ14. For each [word], in what order are its associated grammar
     rules applied,if at all?

Not in scope.
```

Appendix B

Survey materials

Table B.1: Names and assigned identifiers (ID) of the resulting isiXhosa GaliWeather templates. The template names are all suffixed with "Template", however, that is omitted from the table for brevity.

Name	cloud1	cloud2	cloud3	cloud4	cloudc1	cloudc2	cloudc3	temp1
ID	1	2	3	4	5	6	7	9
Name	temp2	temp3	temp4	temp5	temp6	temp7	temp8	temp9
ID	10	11	12	13	14	15	16	17
Name	temp10	temp11	temp12	wind1	wind2	wind3	-	-
ID	18	19	20	21	22	23	-	-

List of texts evaluated in the isiXhosa GaliWeather survey:

```
T0: Kuzakubakho izulu elisibakeleyo ekupheleni, elithe gqabagqaba
     ngamafu esiphakathini kunye nelimathumb' antaka ekupheleni kwemini
T1: Kuzakubakho izulu elisibakeleyo jikelele imini yonke
T2: Kuzakubakho izulu elisibakeleyo ekuqaleni nasesiphakathini semini,
     nangona liyakujika kubekho elimathumb' antaka ekupheleni
T4: Kuzakubakho izulu elisibakeleyo ekuqaleni kwemini, nangona
     liyakujika kubekho elithe gqabagqaba ngamafu ekupheleni
     nasesiphakathini
T5: Kuzakubakho izulu elihle kulemini izayo, nangona ngamanye amaxesha
     kuzakubakho izulu elimathumb' antaka
T6: Silindele utshintsho-ntsitshwano lwezulu elisibakeleyo nelihle
     kwezintsuku zizayo, nangona emaxesheni athile kuzakubakho izulu
     elithe gqabagqaba ngamafu
T7: Kuzakubakho izulu elihle kubekho namaxesha apho kukho izulu
     elimathumb' antaka okanye elisibakeleyo
T8: Imo yesibhakabhaka izokuhluka kakhulu kwimini yonke
T9: Iqondo eliphantsi lemozulu liphezulu kakhulu kwaye neqondo
     eliphezulu liphantsi kakhulu xa lithelekiswa netempritsha
     elindelekileyo kwelixesha enyakeni, kodwa ndawo yonke itemprisha
     ihla ngamandla
T10: Iqondo eliphantsi lemozulu liphantsi kakhulu kwaye neqondo
```

```
     eliphezulu liphantsi kakhulu xa lithelekiswa netempritsha
     elindelekileyo kwelixesha
     enyakeni, kodwa ndawo yonke itemprisha izakunyuka isehla
T11: Iqondo eliphantsi lemozulu liphantsi kakhulu kwaye iqondo
     eliphezulu liphantsi kakhulu xa lithelekiswa nethempristha
     elindelekileyo kwelixesha enyakeni, iqondo eliphantsi lona
     linyuka ngamandla ekuhambeni kwexesha
T12: Iqondo eliphantsi lemozulu liphezulu kakhulu kwaye iqondo
     eliphezulu liphantsi kakhulu xa lithelekiswa nethempritsha
     elindelekileyo kwelixesha
     enyakeni, nangona amaqondo azakunyuka esehla ekuhambeni kwexesha
T13: Iqondo eliphantsi lemozulu liphantsi kakhulu kwaye iqondo
     eliphezulu liphezulu kakhulu xa lithelekiswa nethempristha
     elindelekileyo kwelixesha enyakeni, kwaye iqondo eliphezulu
     linyuka kancinci-kancinci ekuhambeni kwexesha
T14: Iqondo eliphantsi lemozulu liphezulu kakhulu kwaye iqondo
     eliphezulu liphantsi kakhulu xa lithelekiswa nethempristha
     elindelekileyo kwelixesha enyakeni, nangona amaqondo azakunyuka
     esehla
T15: Amaqondo obushushu azakuba ephantsi kakhulu ngelixesha enyakeni,
     kodwa ndawo yonke itemprisha inyuka ngamandla
T16: Amaqondo obushushu azakuba ephantsi kakhulu ngelixesha enyakeni,
     nangona amaqondo azakunyuka esehla
T17: Amaqondo obushushu azakuba ephezulu kakhulu ngelixesha enyakeni,
     iqondo eliphantsi lona lihla kancinci
T18: Amaqondo obushushu azakuba ephantsi kakhulu ngelixesha enyakeni,
     nangona amaqondo awazokunyuka esehla
T19: Amaqondo obushushu azakuba ephantsi kakhulu ngelixesha enyakeni,
     kwaye amaqondo aphezulu anyuka kancinci ekuhambeni kwexesha
T20: Amaqondo obushushu azakuba ephantsi kakhulu ngelixesha enyakeni,
     nangona amaqondo awazokunyuka esehla ekuhambeni kwexesha
T21: Silindele umoya onamandla amakhulu ukusuka emantla ngomvulo
T22: Silindele imimoya ezakuba namandla amakhulu ukusuka emzantsi-mpuma
     ngomvulo
T23: Silindele imimoya ezakuba namandla amakhulu ukusuka emantla-ntshona
     ukuqala ngolwesithathu, itshintse ibe namandla ukusuka emntla-mpuma
     ngolwesihlanu
```

The input to the OWLSIZ verbaliser, its chosen template, and the final text. We use prefix testontology refers to the test ontology taken from Keet and Khumalo [135].

```
Input: SubClassOf(<testonto#ukubhukuda> <testonto#umdlalo>)
Ques. template: Ingabe [subjC]onke [C1] [subjC][COP][C2]?
Output: Ingabe konke ukubhukuda kungumdlalo?

Input: SubClassOf(<testonto#indlovu> <testonto#isilwane>)
Ques. template: Ingabe [subjC]onke [C1] [subjC][COP][C2]?
Output: Ingabe yonke indlovu iyisilwane?

Input: SubClassOf(<testonto#i-okhestra>
     ObjectSomeValuesFrom(<testonto#aaaa> <testonto#isazi_somnyuziki>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
```

```
Output: Ingabe lonke i-okhestra linesazi somnyuziki?

Input: SubClassOf(<testonto#indlovu>
     ObjectSomeValuesFrom(<testonto#dla> <testonto#ihlamvana>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe yonke indlovu idla ihlamvana?

Input: SubClassOf(<testonto#isisu>
     ObjectSomeValuesFrom(<testonto#aaaa> <testonto#indilinga_yokudla>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe sonke isisu sinendilinga yokudla?

Input: SubClassOf(<testonto#ixhiba>
     ObjectSomeValuesFrom(<testonto#dddd> <testonto#iziko>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe lonke ixhiba lineziko?

Input: SubClassOf(<testonto#ibhubesi> <testonto#isilwane>)
Ques. template: Ingabe [subjC]onke [C1] [subjC][COP][C2]?
Output: Ingabe lonke ibhubesi liyisilwane?


Input: SubClassOf(<testonto#ukusebenza>
     ObjectSomeValuesFrom(<testonto#yawufeza>
     <testonto#umsebenzi_onqunyiwe>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe konke ukusebenza kuyawufeza umsebenzi onqunyiwe?

Input: SubClassOf(<testonto#iqembu_labahlinzi>
     ObjectSomeValuesFrom(<testonto#aaaa> <testonto#udokotela>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe lonke iqembu labahlinzi linodokotela?

Input: SubClassOf(<testonto#impala> <testonto#isilwane>)
Ques. template: Ingabe [subjC]onke [C1] [subjC][COP][C2]?
Output: Ingabe yonke impala iyisilwane?

Input: SubClassOf(<testonto#inhliziyo>
     ObjectMinCardinality(1 <testonto#enza> <testonto#ifoni>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe yonke inhliziyo yenza ifoni elingaphezulu kuka-1?

Input: SubClassOf(<testonto#udokotela>
     ObjectSomeValuesFrom(<testonto#ingxenye> <testonto#ukuhlinza>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe wonke udokotela uyingxenye ekuhlinzeni?

Input: SubClassOf(<testonto#isihlahla> <testonto#umuthi>)
Ques. template: Ingabe [subjC]onke [C1] [subjC][COP][C2]?
Output: Ingabe sonke isihlahla singumuthi?

Input: SubClassOf(<testonto#ukugwinya>
     ObjectSomeValuesFrom(<testonto#ingxenye> <testonto#ukudla>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
```

```
Output: Ingabe konke ukugwinya kuyingxenye ekudleni?

Input: SubClassOf(<testonto#iNgqungquthela>
     ObjectSomeValuesFrom(<testonto#aaaa> <testonto#uMnumzana>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe yonke iNgqungquthela inoMnumzana?

Input: SubClassOf(<testonto#inhliziyo>
     ObjectSomeValuesFrom(<testonto#ingxenye> <testonto#umuntu>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe yonke inhliziyo iyingxenye kumuntu?

Input: SubClassOf(<testonto#incwadi>
     ObjectSomeValuesFrom(<testonto#ffff> <testonto#imvilophu>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe yonke incwadi isemvilophini?

Input: SubClassOf(<testonto#umfundi_ongenaziqu>
     ObjectSomeValuesFrom(<testonto#enza> <testonto#isifundo>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe wonke umfundi ongenaziqu wenza isifundo?

Input: SubClassOf(<testonto#umqondo_womshini>
     ObjectSomeValuesFrom(<testonto#ffff> <testonto#ikhompyutha>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe wonke umqondo womshini usekhompyutheni?

Input: SubClassOf(<testonto#ukuhlinza>
     ObjectSomeValuesFrom(<testonto#bbbb> <testonto#iqembu_labahlinzi>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe konke ukuhlinza kuneqembu labahlinzi?

Input: SubClassOf(<testonto#umakhalekhukhwini> <testonto#ifoni>)
Ques. template: Ingabe [subjC]onke [C1] [subjC][COP][C2]?
Output: Ingabe wonke umakhalekhukhwini uyifoni?

Input: SubClassOf(<testonto#indlulamithi>
     ObjectSomeValuesFrom(<testonto#dla> <testonto#ihlamvana>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe yonke indlulamithi idla ihlamvana?

Input: SubClassOf(<testonto#ubabekazi>
     ObjectMaxCardinality(1 <testonto#enza> <testonto#ifoni>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe wonke ubabekazi wenza ifoni elimbalwa ku-1?

Input: SubClassOf(<testonto#indlulamithi> <testonto#isilwane>)
Ques. template: Ingabe [subjC]onke [C1] [subjC][COP][C2]?
Output: Ingabe yonke indlulamithi iyisilwane?

Input: SubClassOf(<testonto#isazi_somnyuziki>
     ObjectSomeValuesFrom(<testonto#ingxenye> <testonto#i-okhestra>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe sonke isazi somnyuziki siyingxenye e-okhestreni?
```

```
120
121 Input: SubClassOf(<testonto#ugogo>
122      ObjectComplementOf(ObjectSomeValuesFrom(<testonto#dla>
123      <testonto#i-apula>)))
124
125 Input: SubClassOf(<testonto#inkawu>
126      ObjectSomeValuesFrom(<testonto#dla> <testonto#isithelo>))
127 Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
128 Output: Ingabe yonke inkawu idla isithelo?
129
130 Input: SubClassOf(<testonto#ufulawa>
131      ObjectSomeValuesFrom(<testonto#ingxenye> <testonto#isinkwa>))
132 Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
133 Output: Ingabe wonke ufulawa uyingxenye esinkweni?
134
135 Input: SubClassOf(<testonto#imvilophu>
136      ObjectSomeValuesFrom(<testonto#aaaa> <testonto#incwadi>))
137 Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
138 Output: Ingabe yonke imvilophu inencwadi?
139
140 Input: SubClassOf(<testonto#isibhedlela>
141      ObjectSomeValuesFrom(<testonto#dddd>
142      <testonto#ithiyetha_yokuhlinzela>))
143 Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
144 Output: Ingabe sonke isibhedlela sinethiyetha yokuhlinzela?
145
146 Input: SubClassOf(<testonto#ukhetho>
147      ObjectSomeValuesFrom(<testonto#bbbb> <testonto#umphakathi>))
148 Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
149 Output: Ingabe lonke ukhetho lunomphakathi?
150
151 Input: SubClassOf(<testonto#umuntu>
152      ObjectSomeValuesFrom(<testonto#aaaa> <testonto#inhliziyo>))
153 Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
154 Output: Ingabe wonke umuntu unenhliziyo?
155
156 Input: SubClassOf(<testonto#indilinga_yokudla>
157      ObjectSomeValuesFrom(<testonto#ffff> <testonto#isisu>))
158 Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
159 Output: Ingabe yonke indilinga yokudla isesiswini?
160
161 Input: SubClassOf(<testonto#ivazi>
162      ObjectSomeValuesFrom(<testonto#akhiwe> <testonto#ubumba>))
163 Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
164 Output: Ingabe yonke ivazi yakhiwe ubumba?
165
166 Input: SubClassOf(<testonto#ingwe> <testonto#isilwane>)
167 Ques. template: Ingabe [subjC]onke [C1] [subjC][COP][C2]?
168 Output: Ingabe yonke ingwe iyisilwane?
169
170 Input: SubClassOf(<testonto#isifundo>
171      ObjectSomeValuesFrom(<testonto#fundiswa> <testonto#uSolwazi>))
172 Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
173 Output: Ingabe sonke isifundo sifundiswa uSolwazi?
```

```
174
175 Input: SubClassOf(<testonto#iziko>
176      ObjectSomeValuesFrom(<testonto#umunxa> <testonto#ixhiba>))
177 Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
178 Output: Ingabe lonke iziko lumunxa ixhiba?
179
180 Input: SubClassOf(<testonto#indlu>
181      ObjectSomeValuesFrom(<testonto#akhiwe> <testonto#itshe>))
182 Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
183 Output: Ingabe yonke indlu yakhiwe itshe?
184
185 Input: SubClassOf(<testonto#iqembu_labahlinzi>
186      ObjectSomeValuesFrom(<testonto#hlanganyele> <testonto#ukuhlinza>))
187 Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
188 Output: Ingabe lonke iqembu labahlinzi lihlanganyele ukuhlinza?
189
190 Input: SubClassOf(<testonto#uMnumzana>
191      ObjectSomeValuesFrom(<testonto#ingxenye> <testonto#iNgqungquthela>))
192 Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
193 Output: Ingabe wonke uMnumzana uyingxenye eNgqungqutheleni?
194
195 Input: SubClassOf(<testonto#ithiyetha_yokuhlinzela>
196      ObjectSomeValuesFrom(<testonto#umunxa> <testonto#isibhedlela>))
197 Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
198 Output: Ingabe yonke ithiyetha yokuhlinzela umunxa isibhedlela?
199
200 Input: SubClassOf(<testonto#ukuhlinza>
201      ObjectSomeValuesFrom(<testonto#aaaa> <testonto#udokotela>))
202 Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
203 Output: Ingabe konke ukuhlinza kunodokotela?
204
205 Input: SubClassOf(<testonto#ikhompyutha>
206      ObjectSomeValuesFrom(<testonto#aaaa> <testonto#umqondo_womshini>))
207 Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
208 Output: Ingabe lonke ikhompyutha linomqondo womshini?
209
210 Input: SubClassOf(<testonto#ihebhu> <testonto#umuthi>)
211 Ques. template: Ingabe [subjC]onke [C1] [subjC][COP][C2]?
212 Output: Ingabe lonke ihebhu lingumuthi?
213
214 Input: SubClassOf(<testonto#umshobingo>
215      ObjectSomeValuesFrom(<testonto#cccc> <testonto#amanzi>))
216 Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
217 Output: Ingabe wonke umshobingo unamanzi?
218
219 Input: SubClassOf(<testonto#inkawu> <testonto#isilwane>)
220 Ques. template: Ingabe [subjC]onke [C1] [subjC][COP][C2]?
221 Output: Ingabe yonke inkawu iyisilwane?
222
223 Input: SubClassOf(<testonto#iphilisi>
224      ObjectSomeValuesFrom(<testonto#enziwe> <testonto#isitashi>))
225 Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
226 Output: Ingabe lonke iphilisi lenziwe isitashi?
227
```

```
Input: SubClassOf(<testonto#umuntu>
     ObjectSomeValuesFrom(<testonto#dla> <testonto#isithelo>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe wonke umuntu udla isithelo?

Input: SubClassOf(<testonto#umphakathi>
     ObjectSomeValuesFrom(<testonto#hlanganyele> <testonto#ukhetho>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe wonke umphakathi uhlanganyele ukhetho?

Input: SubClassOf(<testonto#inja> <testonto#isilwane>)
Ques. template: Ingabe [subjC]onke [C1] [subjC][COP][C2]?
Output: Ingabe yonke inja iyisilwane?

Input: SubClassOf(<testonto#umfundi_ongenaziqu> <testonto#umfundi>)
Ques. template: Ingabe [subjC]onke [C1] [subjC][COP][C2]?
Output: Ingabe wonke umfundi ongenaziqu ungumfundi?

Input: SubClassOf(<testonto#umuntu>
     ObjectSomeValuesFrom(<testonto#phuza> <testonto#uketshezi>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe wonke umuntu uphuza uketshezi?

Input: SubClassOf(<testonto#ikhambi> <testonto#umuthi>)
Ques. template: Ingabe [subjC]onke [C1] [subjC][COP][C2]?
Output: Ingabe lonke ikhambi lingumuthi?

Input: SubClassOf(<testonto#isinkwa>
     ObjectSomeValuesFrom(<testonto#cccc> <testonto#ufulawa>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe sonke isinkwa sinofulawa?

Input: SubClassOf(<testonto#ibhubesi>
     ObjectSomeValuesFrom(<testonto#dla> <testonto#impala>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe lonke ibhubesi lidla impala?


Input: SubClassOf(<testonto#ukudla>
     ObjectSomeValuesFrom(<testonto#aaaa> <testonto#ukugwinya>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe konke ukudla kunokugwinya?

Input: SubClassOf(<testonto#ibhotela>
     ObjectExactCardinality(1 <testonto#enza> <testonto#ifoni>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe lonke ibhotela lenza ifoni eliyi-1 ncamashi?

Input: SubClassOf(<testonto#amanzi>
     ObjectSomeValuesFrom(<testonto#ingxenye> <testonto#umshobingo>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe onke amanzi ayingxenye emshobingweni?

Input: SubClassOf(<testonto#ihlamvana>
```

```
     ObjectAllValuesFrom(<testonto#ingxenye> <testonto#isihlalo>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe lonke ihlamvana lingxenye isihlalo sodwa?

Input: SubClassOf(<testonto#uSolwazi>
     ObjectSomeValuesFrom(<testonto#fundisa> <testonto#isifundo>))
Ques. template: Ingabe [subjC]onke [C1] [subjC][C2]?
Output: Ingabe wonke uSolwazi ufundisa isifundo?

Input: DataPropertyAssertion(<testonto#neminyaka>
     <testonto#uZola> "50"^^xsd:int)
Ques. template: Ingabe [I] [DP] [L]?
Output: Ingabe uZola neminyaka 50?

Input: DisjointClasses(<testonto#ibhubesi> <testonto#indlovu>)
Ques. template: a[subjCon]kho yini [C1] [relCon][COP][C2]?
Output: alikho yini ibhubesi eliyindlovu?

Input: DisjointClasses(<testonto#indoda> <testonto#umfazi>)
Ques. template: a[subjCon]kho yini [C1] [relCon][COP][C2]?
Output: alikho yini indoda elingumfazi?

Input: DisjointClasses(<testonto#isidlanyama> <testonto#isidlazitshalo>)
Ques. template: a[subjCon]kho yini [C1] [relCon][COP][C2]?
Output: asikho yini isidlanyama esiyisidlazitshalo?

Input: DisjointClasses(<testonto#indebe> <testonto#ingilazi>)
Ques. template: a[subjCon]kho yini [C1] [relCon][COP][C2]?
Output: ayikho yini indebe eyingilazi?

Input: DisjointClasses(<testonto#ukugijima> <testonto#ukuhlala>)
Ques. template: a[subjCon]kho yini [C1] [relCon][COP][C2]?
Output: akukho yini ukugijima okungukuhlala?

Input: DisjointClasses(<testonto#ibhotela> <testonto#ushizi>)
Ques. template: a[subjCon]kho yini [C1] [relCon][COP][C2]?
Output: alikho yini ibhotela elingushizi?

Input: DisjointClasses(<testonto#ubabekazi> <testonto#umalume>)
Ques. template: a[subjCon]kho yini [C1] [relCon][COP][C2]?
Output: awukho yini ubabekazi ungumalume?

Input: DisjointClasses(<testonto#ihebhivo> <testonto#ikhanivo>)
Ques. template: a[subjCon]kho yini [C1] [relCon][COP][C2]?
Output: alikho yini ihebhivo eliyikhanivo?

Input: DisjointClasses(<testonto#indlu> <testonto#umjondolo>)
Ques. template: a[subjCon]kho yini [C1] [relCon][COP][C2]?
Output: ayikho yini indlu engumjondolo?

Input: DisjointClasses(<testonto#inyama> <testonto#umfino>)
Ques. template: a[subjCon]kho yini [C1] [relCon][COP][C2]?
Output: ayikho yini inyama engumfino?

```

```
Input: DisjointClasses(<testonto#isipunu> <testonto#umfoloko>)
Ques. template: a[subjCon]kho yini [C1] [relCon][COP][C2]?
Output: asikho yini isipunu esingumfoloko?

Input: DisjointClasses(<testonto#isihlalo> <testonto#itafula>)
Ques. template: a[subjCon]kho yini [C1] [relCon][COP][C2]?
Output: asikho yini isihlalo esiyitafula?

Input: DisjointClasses(<testonto#umfula> <testonto#umsele>)
Ques. template: a[subjCon]kho yini [C1] [relCon][COP][C2]?
Output: awukho yini umfula ongumsele?

Input: ObjectPropertyAssertion(<testonto#fundisa>
     <testonto#iNokia_3310> <testonto#uZola>)
Ques. template: [I1] [subjCon][OP] [I2]?
Output: iNokia 3310 lifundisa uZola?

Input: EquivalentClasses(<testonto#indlu> <testonto#inyama> )
Ques. template: Ingabe noma yi[objCon]phi [C2] [relCon][COP][C1]?
Output: Ingabe noma yiyiphi indlu eyinyama?

Input: ClassAssertion(<testonto#umakhalekhukhwini>
     <testonto#iNokia_3310>)
Ques. template: Ingabe [I] [subjCon][COP][C]?
Output: Ingabe iNokia 3310 lingumakhalekhukhwini?

Input: ClassAssertion(<testonto#indoda> <testonto#uZola>)
Ques. template: Ingabe [I] [subjCon][COP][C]?
Output: Ingabe uZola uyindoda?
```

Quality of computer produced descriptions of the weather

Invitation to participate, and benefits: You are invited to participate in a research study conducted by members of the Knowledge Engineering team (www.meteck.org/keen/) from the Computer Science department at the University of Cape Town, South Africa. The study aim is to determine the difference in naturalness and quality of utterances produced by data-driven and template-based text generation systems. We believe that your experience would be a valuable source of information, and hope that by participating you may gain useful knowledge.

Procedures: During this study, you will be asked to rate isiXhosa texts based on their fluency and grammaticality.

Recording: We will not take photographs and/or record audio/video as part of the study.

Risks: There are no potentially harmful risks related to your participation in this study.

Disclaimer/Withdrawal: Your participation is completely voluntary; you may refuse to participate, and you may withdraw at any time without having to state a reason and without any prejudice or penalty against you. Should you choose to withdraw, the researchers commit not to use any of the information you have provided without your signed consent. Note also that the researchers may withdraw you from the study at any time.

Confidentiality: All information collected in this study will be kept private in that you will not be identified by name or by affiliation to an institution. Confidentiality and anonymity will be maintained as pseudonyms will be used.

Ethics approval code: FSREC 014-2020

What signing this form means: By clicking on the "Next" button, you agree to participate in this research study. The aim, procedures to be used, as well as the potential risks and benefits of your participation have been explained to you in detail, using this form. Refusal to participate in or withdrawal from this study at any time will have no effect on you in any way. You are free to contact us (via zmahlaza@cs.uct.ac.za), to ask questions or request further information, at any time during this research.

* Required

Figure B.1: English instructions that were available to participants in the IsiXhosa GaliWeather survey

Umgangatho wengxelo yemozulu ebhalwe yikhompyutha

Isimemo sokuthatha inxaxheba, kunye nezibonelelo: Uyamenywa ukuba uthathe inxaxheba kuphando olwenziwe ngamalungu eqela le Knowledge Engineering (www.meteck.org/keen/) kwisebe lezeNzululwazi ngeKhompyutha kwiYunivesithi yaseKapa, eMzantsi Afrika. Injongo yoluphando kukufumanisa umgangatho wemibhalo eyenziwa yisistimu yokwenza ulwimi lwendalo. Sikholelwa ukuba amava akho angaba ngumthombo wolwazi oluxabisekileyo, kwaye sinethemba lokuba ngokuthatha inxaxheba unokufumana ulwazi olululuncedo.

Inkqubo: Koluphando, uzakucelwa ukuba uthelekelele ubuciko kunye nokugobela imigaqo mthetho yegrama kwimibhalo yesiXhosa ozakube uyinikiwe.

Ukurekhoda: Asizokuthatha iifoto kunye/okanye ukurekhoda isandi/ividiyo njengenxalenye yesifundo.

Ubungozi: Akukho mingcipheko inokuba yingozi ngokunxulumene nenxaxheba yakho kolu phando.

Ukurhoxa: Uthatho-nxaxheba kuxhomekeke kwintando yakho ngokupheleleyo; unokwala ukuthatha inxaxheba, kwaye ungarhoxa nangaliphi ixesha ungakhange uchaze sizathu kwaye akuzokubakho isigwebo okanye isohlwayo ngokuchasene nawe ngenxa yokurhoxa. Ukuba ukhetha ukurhoxa, abaphandi bayazibophelela ekubeni abazokulisebenzisa ulwazi obanike lona ngaphandle kwemvume yakho esayiniweyo. Qaphela kwakhona ukuba abaphandi banokurhoxisa esifundweni nangaliphi na ixesha.

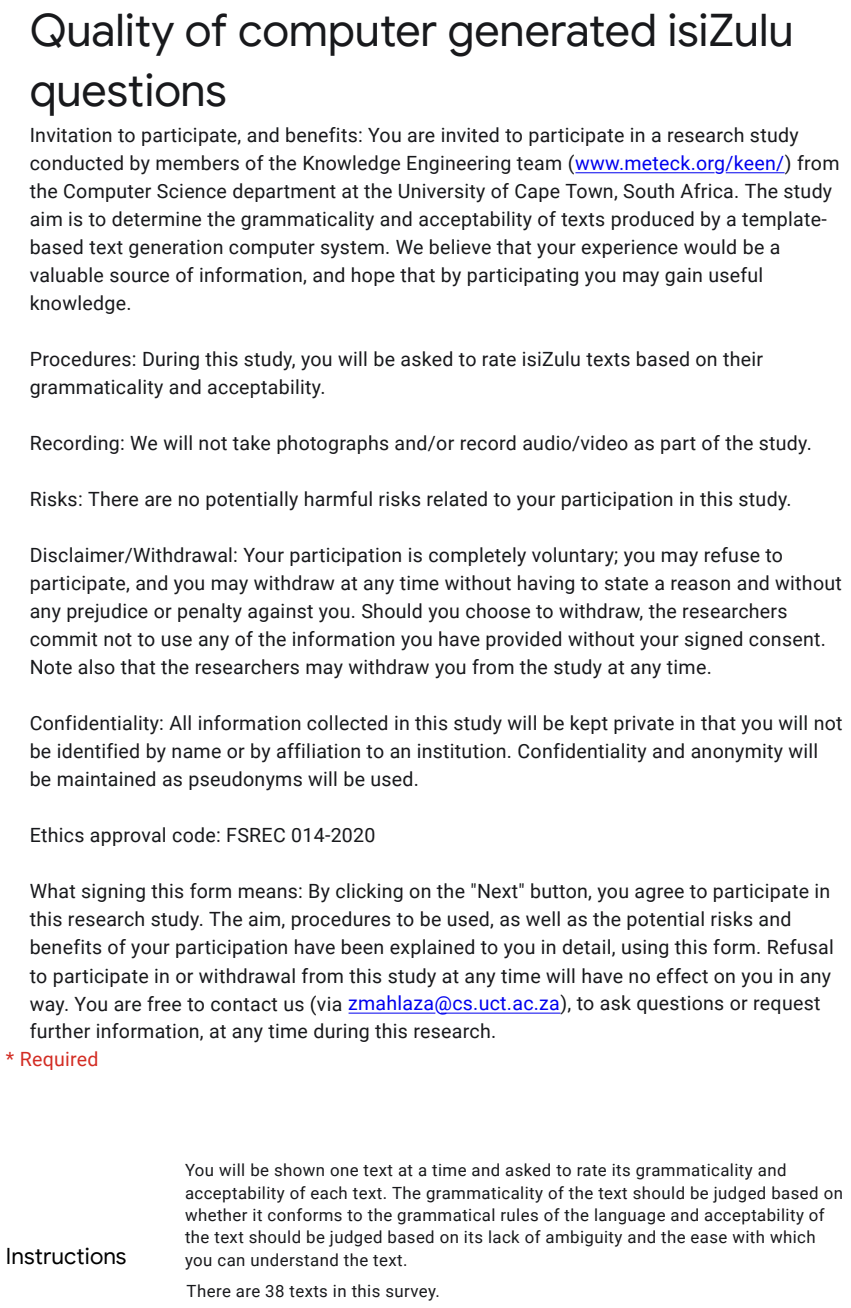
Ubumfihlo: Lonke ulwazi oluqokelelwe koluphando luyakugcinwa luyimfihlo njengokuba awuzokuchongwa ngegama okanye ngonxibelelwano lwakho neziko ethile. Izinto eziyimfihlo kunye nenkcukacha zakho zisakufihlwa ngoba kuzakusetyenziswa igama elingelilo elakho.

Ikhowudi yokuvunywa kokuziphatha: FSREC 014-2020

Kuthetha ntoni ukusayina le fomu: Ngokucofa iqhosha elibhalwe "Next", uyavuma ukuthatha inxaxheba koluphando. Injongo, imigaqo-nkqubo eza kusetyenziswa, kunye nobungozi obunokubakho kunye nezibonelelo zokuthatyathwa kwenxaxheba zichaziwe kuwe ngokucacileyo kule fomu. Ukwala ukubandakanyeka okanye ukurhoxa kolu phando ngalo naliphi na ixesha akusayi kuba nempembelelo nangayiphi na indlela. Ukhululekile ukunxibelelana nathi (via zmahlaza@cs.uct.ac.za), ukubuza imibuzo okanye ukucela ulwazi oluthe kratya, ngalo naliphi na ixesha loluphando.

* Required

Figure B.2: IsiXhosa instructions that were available to participants in the IsiXhosa GaliWeather survey



Quality of computer generated isiZulu questions

Invitation to participate, and benefits: You are invited to participate in a research study conducted by members of the Knowledge Engineering team (www.meteck.org/keen/) from the Computer Science department at the University of Cape Town, South Africa. The study aim is to determine the grammaticality and acceptability of texts produced by a template-based text generation computer system. We believe that your experience would be a valuable source of information, and hope that by participating you may gain useful knowledge.

Procedures: During this study, you will be asked to rate isiZulu texts based on their grammaticality and acceptability.

Recording: We will not take photographs and/or record audio/video as part of the study.

Risks: There are no potentially harmful risks related to your participation in this study.

Disclaimer/Withdrawal: Your participation is completely voluntary; you may refuse to participate, and you may withdraw at any time without having to state a reason and without any prejudice or penalty against you. Should you choose to withdraw, the researchers commit not to use any of the information you have provided without your signed consent. Note also that the researchers may withdraw you from the study at any time.

Confidentiality: All information collected in this study will be kept private in that you will not be identified by name or by affiliation to an institution. Confidentiality and anonymity will be maintained as pseudonyms will be used.

Ethics approval code: FSREC 014-2020

What signing this form means: By clicking on the "Next" button, you agree to participate in this research study. The aim, procedures to be used, as well as the potential risks and benefits of your participation have been explained to you in detail, using this form. Refusal to participate in or withdrawal from this study at any time will have no effect on you in any way. You are free to contact us (via zmahlaza@cs.uct.ac.za), to ask questions or request further information, at any time during this research.

* Required

Instructions

You will be shown one text at a time and asked to rate its grammaticality and acceptability of each text. The grammaticality of the text should be judged based on whether it conforms to the grammatical rules of the language and acceptability of the text should be judged based on its lack of ambiguity and the ease with which you can understand the text.

There are 38 texts in this survey.

Figure B.3: English instructions that were available to participants in the OWLSIZ survey

UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

Faculty of Science
University of Cape Town
Rondebosch
South Africa 7701

E-mail: shari.daya@uct.ac.za
Tel: 021 650-2880

2 April 2020

Mr. Zola Mahlaza
Department of Computer Science

The evaluation of the quality of isiXhosa weather forecast and isiZulu ontology verbalisation text

Dear Mr. Zola Mahlaza

I am pleased to inform you that the Faculty of Science Research Ethics Committee has approved the above-named application for research ethics clearance, subject to the conditions listed below.

- Restrictions on involving human participants in research must be adhered to, given current concerns about the spread of Covid-19. Please ensure that you are aware of and comply with UCT policy on this, as communicated by management.
- Implement the measures described in your application to ensure that the process of your research is ethically sound; and
- Uphold ethical principles throughout all stages of the research, responding appropriately to unanticipated issues: please contact me if you need advice on ethical issues that arise.

Your approval code is: **FSREC 014- 2020**

I wish you success in your research.

Yours sincerely

Dr Shari Daya
Chair: Faculty of Science Research Ethics Committee

Cc: **A/Prof. Maria Keet (Supervisor)**

Figure B.4: Ethics approval document for the evaluation of the quality of isiXhosa weather forecast and isiZulu ontology verbalisation text

Bibliography

[1] D. M. Abdullah and Y. Chali. "Towards Generating Query to Perform Query Focused Abstractive Summarization using Pre-trained Model". In: *Proceedings of the 13th International Conference on Natural Language Generation, INLG 2020, Dublin, Ireland, December 15-18, 2020*. Ed. by B. Davis, Y. Graham, J. D. Kelleher, and Y. Sripada. Association for Computational Linguistics, 2020, pp. 80–85.

[2] W. Abed and E. Reiter. "Arabic NLG Language Functions". In: *Proceedings of the 13th International Conference on Natural Language Generation, INLG 2020, Dublin, Ireland, December 15-18, 2020*. Association for Computational Linguistics, 2020, pp. 7–14.

[3] G. Aguado, A. Bañón, J. Bateman, S. Bernardos, M. Fernández, A. Gómez-Pérez, E. Nieto, A. Olalla, R. Plaza, and A. Sánchez. "ONTOGENERATION: Reusing domain and linguistic ontologies for Spanish text generation". In: *Workshop on Applications of Ontologies and Problem Solving Methods, 13th European Conference on Artificial Intelligence, August 23-28 1998, Brighton, UK*.

[4] I. Akermi, J. Heinecke, and F. Herledan. "Tansformer based Natural Language Generation for Question-Answering". In: *Proceedings of the 13th International Conference on Natural Language Generation, INLG 2020, Dublin, Ireland, December 15-18, 2020*. Ed. by B. Davis, Y. Graham, J. D. Kelleher, and Y. Sripada. Association for Computational Linguistics, 2020, pp. 349–359.

[5] M. M. Al-Yahya. "OntoQue: A Question Generation Engine for Educational Assesment Based on Domain Ontologies". In: *Proceedings of the 11th IEEE International Conference on Advanced Learning Technologies, ICALT 2011, Athens, Georgia, USA, 6-8 July 2011*. IEEE Computer Society, 2011, pp. 393–395.

[6] N. Alexander. "The political economy of the harmonisation of the Nguni and the Sotho languages". In: *Lexikos* 8.8 (1998), pp. 269–275.

[7] M. Amith, F. J. Manion, M. R. Harris, Y. Zhang, H. Xu, and C. Tao. "Expressing Biomedical Ontologies in Natural Language for Expert Evaluation". In: *Proceedings of the 16th World Congress on Medical and Health Informatics, MEDINFO 2017: Precision Healthcare through Informatics, Hangzhou, China, 21-25 August 2017*. Ed. by A. V. Gundlapalli, M. Jaulent, and D.

Zhao. Vol. 245. Studies in Health Technology and Informatics. IOS Press, 2017, pp. 838–842.

[8] I. Androutsopoulos, G. Lampouras, and D. Galanis. "Generating Natural Language Descriptions from OWL Ontologies: the NaturalOWL System". In: *Journal of Artificial Intelligence Research* 48 (2013), pp. 671–715.

[9] W. T. Ang, R. Kanagasabai, and C. J. O. Baker. "Knowledge Translation: Computing the Query Potential of Bio-ontologies". In: *Proceedings of the Workshop on Semantic Web Applications and Tools for Life Sciences (SWAT4LS), Edinburgh, United Kingdom, November 28, 2008*. Ed. by A. Burger, A. Paschke, P. Romano, and A. Splendiani. Vol. 435. CEUR Workshop Proceedings. CEUR-WS.org, 2008.

[10] G. Angeli, P. Liang, and D. Klein. "A Simple Domain-Independent Probabilistic Approach to Generation". In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, EMNLP 2010, 9-11 October 2010, MIT Stata Center, Massachusetts, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2010, pp. 502–512.

[11] K. Angelov and R. Enache. "Typeful Ontologies with Direct Multilingual Verbalization". In: *Proceedings of the Second International Workshop on Controlled Natural Language, CNL 2010, Revised Papers*. Ed. by M. Rosner and N. E. Fuchs. Vol. 7175. Lecture Notes in Computer Science. September 13-15, Marettimo Island, Italy. Springer, 2010, pp. 1–20.

[12] D. E. Appelt. "Planning English Referring Expressions". In: *Artificial Intelligence* 26.1 (1985), pp. 1–33.

[13] D. Bahdanau, K. Cho, and Y. Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. 2015.

[14] J. Baldridge and G. M. Kruijff. "Coupling CCG and Hybrid Logic Dependency Semantics". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*. ACL, 2002, pp. 319–326.

[15] D. Bamutura and P. Ljunglöf. "Towards a Resource Grammar for Runyankore and Rukiga". In: *Proceedings of the 2019 Workshop on Widening NLP*. Florence, Italy: Association for Computational Linguistics, Aug. 2019, pp. 2–6.

[16] V. Basile and A. Mazzei. "The DipInfo-UniTo system for SRST 2018". In: *Proceedings of the First Workshop on Multilingual Surface Realisation*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 65–71.

[17] J. A. Bateman, R. T. Kasper, J. D. Moore, and R. A. Whitney. *A general organization of knowledge for natural language processing: the Penman upper model*. Tech. rep. Information Sciences Institute, Univeristy of Southern California, Marina Del Rey, USA, 1990.

[18] J. A. Bateman. “Enabling technology for multilingual natural language generation: the KPML development environment”. In: *Natural Language Engineering* 3.1 (1997), pp. 15–55.

[19] S. Bechhofer, F. Van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. *OWL web ontology language reference*. URL: `https://www.w3.org/TR/owl-ref/#FunctionalProperty-def`.

[20] A. Belz. “Probabilistic Generation of Weather Forecast Texts”. In: *Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, April 22-27, 2007, Rochester, New York, USA*. Ed. by C. L. Sidner, T. Schultz, M. Stone, and C. Zhai. Association for Computational Linguistics, 2007, pp. 164–171.

[21] A. Belz. “Automatic generation of weather forecast texts using comprehensive probabilistic generation-space models”. In: *Natural Language Engineering* 14.4 (2008), pp. 431–455.

[22] A. Belz, B. Bohnet, S. Mille, L. Wanner, and M. White. “The Surface Realisation Task: Recent Developments and Future Plans”. In: *INLG 2012 - Proceedings of the Seventh International Natural Language Generation Conference, 30 May 2012 - 1 June 2012, Starved Rock State Park, Utica, IL, USA*. Ed. by B. D. Eugenio, S. McRoy, A. Gatt, A. Belz, A. Koller, and K. Striegnitz. The Association for Computer Linguistics, 2012, pp. 136–140.

[23] A. Belz, M. White, D. Espinosa, E. Kow, D. Hogan, and A. Stent. “The First Surface Realisation Shared Task: Overview and Evaluation Results”. In: *ENLG 2011 - Proceedings of the 13th European Workshop on Natural Language Generation, 28-30 September 2011, Nancy, France*. Ed. by C. Gardent and K. Striegnitz. The Association for Computer Linguistics, 2011, pp. 217–226.

[24] A. Belz, S. Mille, and D. M. Howcroft. “Disentangling the Properties of Human Evaluation Methods: A Classification System to Support Comparability, Meta-Evaluation and Reproducibility Testing”. In: *Proceedings of the 13th International Conference on Natural Language Generation, INLG 2020, Dublin, Ireland, December 15-18, 2020*. Ed. by B. Davis, Y. Graham, J. D. Kelleher, and Y. Sripada. Association for Computational Linguistics, 2020, pp. 183–194.

[25] A. Bertolino, G. De Angelis, A. Di Sandro, and A. Sabetta. “Is my model right? Let me ask the expert”. In: *Journal of Systems and Software* 84.7 (2011), pp. 1089–1099.

[26] W. H. I. Bleek. *A comparative grammar of South African languages*. Trübner and Company, 1862.

[27] B. Bohnet, L. Wanner, S. Mille, and A. Burga. “Broad Coverage Multilingual Deep Sentence Generation with a Stochastic Multi-Level Realizer”. In: *Proceedings of the 23rd International Conference on Computational Linguistics, COLING 2010, 23-27 August 2010, Beijing, China*. Tsinghua University Press, 2010, pp. 98–106.

[28] M. Bollmann. “Adapting SimpleNLG to German”. In: *Proceedings of the 13th European Workshop on Natural Language Generation, ENLG 2011, 28-30 September 2011, Nancy, France*. Association for Computational Linguistics, 2011, pp. 133–138.

[29] K. Bontcheva. “Generating Tailored Textual Summaries from Ontologies”. In: *The Semantic Web: Research and Applications, Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005, Proceedings*. Ed. by A. Gómez-Pérez and J. Euzenat. Vol. 3532. Lecture Notes in Computer Science. Springer, 2005, pp. 531–545.

[30] K. Bontcheva and Y. Wilks. “Automatic Report Generation from Ontologies: The MIAKT Approach”. In: *Natural Language Processing and Information Systems, 9th International Conference on Applications of Natural Languages to Information Systems, NLDB 2004, Salford, UK, June 23-25, 2004, Proceedings*. Ed. by F. Meziane and E. Métais. Vol. 3136. Lecture Notes in Computer Science. Springer, 2004, pp. 324–335.

[31] S. E. Bosch, T. Eckart, B. Klimek, D. Goldhahn, and U. Quasthoff. “Preparation and Usage of Xhosa Lexicographical Data for a Multilingual, Federated Environment”. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018*. 2018.

[32] N. Bouayad-Agha, G. Casamayor, S. Mille, M. Rospocher, H. Saggion, L. Serafini, and L. Wanner. “From Ontology to NL: Generation of Multilingual User-Oriented Environmental Reports”. In: *Proceedings of the 17th International Conference on Applications of Natural Language to Information Systems, NLDB 2012 - Natural Language Processing and Information Systems, Groningen, The Netherlands, June 26-28, 2012*. Springer, 2012, pp. 216–221.

[33] N. Bouayad-Agha, G. Casamayor, S. Mille, and L. Wanner. “Perspective-oriented generation of football match summaries: Old tasks, new challenges”. In: *ACM Trans. Speech Lang. Process.* 9.2 (2012), 3:1–3:31.

[34] W. Bourquin. “Notes on the concords in Xhosa and Zulu, their differences and general aspect”. In: *African Studies* 11.1 (1952), pp. 16–28.

[35] T. Bouttaz, E. Pignotti, C. Mellish, and P. Edwards. “A Policy-Based Approach to Context Dependent Natural Language Generation”. In: *Proceedings of the 13th European Workshop on Natural Language Generation, ENLG 2011, 28-30 September 2011, Nancy, France*. ACL, 2011, pp. 151–157.

[36] D. Braun, K. Klimt, D. Schneider, and F. Matthes. “SimpleNLG-DE: Adapting SimpleNLG 4 to German”. In: *Proceedings of the 12th International Conference on Natural Language Generation, INLG 2019, Tokyo, Japan, October 29 - November 1, 2019*. Ed. by K. van Deemter, C. Lin, and H. Takamura. Association for Computational Linguistics, 2019, pp. 415–420.

[37] D. Braun, E. Reiter, and A. Siddharthan. “SaferDrive: An NLG-based behaviour change support system for drivers”. In: *Natural Language Engineering* 24.4 (2018), pp. 551–588.

[38] BS ISO/IEC 25010:2011. *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. Standard. London, UK: British Standard Institution, Mar. 2011.

[39] M. L. Burton and L. Kirk. “Semantic reality of Bantu noun classes: the Kikuyu case”. In: *Studies in African Linguistics* 7.2 (July 1976), pp. 157–174.

[40] S. Busemann. “Best-First Surface Realization”. In: *arXiv e-prints* (May 1996). arXiv: `cmp-lg/9605010 [cs.CL]`.

[41] S. Busemann and H. Horacek. “A Flexible Shallow Approach To Text Generation”. In: *Proceedings of the Ninth International Workshop on Natural Language Generation, INLG 1998*. Niagara-on-the-Lake, Ontario, Canada, August 5-7. 1998.

[42] J. Byamugisha. “Ontology verbalization in agglutinating Bantu languages: a study of Runyankore and its generalizability”. PhD thesis. Department of Computer Science, University of Cape Town, South Africa, 2019.

[43] J. Byamugisha, C. M. Keet, and B. DeRenzi. “Evaluation of a Runyankore Grammar Engine for Healthcare Messages”. In: *Proceedings of the 10th International Conference on Natural Language Generation, INLG 2017, Santiago de Compostela, Spain, September 4-7, 2017*. 2017, pp. 105–113.

[44] J. Byamugisha, C. M. Keet, and B. DeRenzi. “Bootstrapping a Runyankore CNL from an isiZulu CNL”. In: *Proceedings of the 5th International Workshop on Controlled Natural Language, CNL 2016, Aberdeen, UK, July 25-27, 2016*. Ed. by B. Davis, G. J. Pace, and A. Z. Wyner. Vol. 9767. Lecture Notes in Computer Science. Springer, 2016, pp. 25–36.

[45] J. Byamugisha, C. M. Keet, and B. DeRenzi. “Tense and Aspect in Runyankore Using a Context-Free Grammar”. In: *Proceedings of the Ninth International Conference on Natural Language Generation, INLG 2016, September 5-8, 2016, Edinburgh, UK*. Association for Computational Linguistics, 2016, pp. 84–88.

[46] J. Byamugisha, C. M. Keet, and B. DeRenzi. “Pluralizing Nouns across Agglutinating Bantu Languages”. In: *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*. Ed. by E. M. Bender, L. Derczynski, and P. Isabelle. Association for Computational Linguistics, 2018, pp. 2633–2643.

[47] J. J. Camilleri, N. E. Fuchs, and K. Kaljurand. *ACE grammar library*. Tech. rep. D11.1. Multilingual Online Translation (MOLTO) project, 2012.

[48] N. N. Canonici. *The grammatical structure of Zulu*. Durban: University of Natal, 1987.

[49] N. N. Canonici. *Elements of Zulu morpho-syntax*. Rev. ed. Durban: Zulu Language & Literature, University of Natal, 1995.

[50] M. A. Casteleiro, J. Des, M. J. F. Prieto, R. Perez, and S. Lekkas. "An Ontology-Based Approach to Natural Language Generation from Coded Data in Electronic Health Records". In: *Proceedings of the UKSim 5th European Symposium on Computer Modeling and Simulation, EMS 2011, Madrid, Spain, November 16-18, 2011*. Ed. by D. Al-Dabass, A. Orsoni, A. A. Pantelous, G. Romero, and J. Félez. IEEE, 2011, pp. 366–371.

[51] T. Castro Ferreira, C. van der Lee, E. van Miltenburg, and E. Krahmer. "Neural data-to-text generation: A comparison between pipeline and end-to-end architectures". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 552–562.

[52] T. Castro Ferreira, S. Wubben, and E. Krahmer. "Surface Realization Shared Task 2018 (SR18): The Tilburg University Approach". In: *Proceedings of the First Workshop on Multilingual Surface Realisation*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 35–38.

[53] V. K. Chaudhri, P. E. Clark, A. Overholtzer, and A. Spaulding. "Question Generation from a Knowledge Base". In: *Proceedings of the 19th International Conference on Knowledge Engineering and Knowledge Management, EKAW 2014*. Ed. by K. Janowicz, S. Schlobach, P. Lambrix, and E. Hyvönen. Vol. 8876. Lecture Notes in Computer Science. November 24-28, Linköping, Sweden. Springer, 2014, pp. 54–65.

[54] C. Chavula and C. M. Keet. "Is *Lemon* Sufficient for Building Multilingual Ontologies for Bantu Languages?" In: *Proceedings of the 11th International Workshop on OWL: Experiences and Directions (OWLED 2014) Co-Located with 13th International Semantic Web Conference on (ISWC 2014), Riva Del Garda, Italy, October 17-18, 2014*. Vol. 1265. CEUR Workshop Proc. CEUR-WS.org, 2014, pp. 61–72.

[55] C. Chavula and C. M. Keet. "An Orchestration Framework for Linguistic Task Ontologies". In: *Metadata and Semantics Research - 9th Research Conference, MTSR 2015, Manchester, UK, September 9-11, 2015, Proc.* 2015, pp. 3–14.

[56] K. Chen, F. Li, B. Hu, W. Peng, Q. Chen, H. Yu, and Y. Xiang. "Neural data-to-text generation with dynamic content planning". In: *Knowl. Based Syst.* 215 (2021), p. 106610.

[57] Z. Chen, H. Eavani, W. Chen, Y. Liu, and W. Y. Wang. "Few-Shot NLG with Pre-Trained Language Model". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*. Ed. by D. Jurafsky, J. Chai, N. Schluter, and J. R. Tetreault. Association for Computational Linguistics, 2020, pp. 183–190.

[58] A. Chisholm, W. Radford, and B. Hachey. “Learning to generate one-sentence biographies from Wikidata”. In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1: Long Papers*. Ed. by M. Lapata, P. Blunsom, and A. Koller. Association for Computational Linguistics, 2017, pp. 633–642.

[59] P. Ciccarese and S. Peroni. “The Collections Ontology: Creating and handling collections in OWL 2 DL frameworks”. In: *Semantic Web* 5.6 (2014), pp. 515–529.

[60] P. Cimiano, J. Lüker, D. Nagel, and C. Unger. “Exploiting Ontology Lexica for Generating Natural Language Texts from RDF Data”. In: *Proceedings of the 14th European Workshop on Natural Language Generation, ENLG 2013, August 8-9, 2013, Sofia, Bulgaria*. Ed. by A. Gatt and H. Saggion. Association for Computer Linguistics, 2013, pp. 10–19.

[61] J. Coch. “Overview of AlethGen”. In: *Eighth International Natural Language Generation Workshop, INLG 1996, Herstmonceux Castle, Sussex, UK, June 12-15, 1996 - Posters and Demonstrations*. 1996.

[62] D. A. Cojocaru and S. Trausan-Matu. “Text Generation Starting from an Ontology”. In: *Proceedings of the 12th Romanian Human-Computer Interaction Conference, RoCHI 2015, Bucharest, Romania, September 24-25, 2015*. Ed. by M. Dardala and T. Rebedea. Matrix Rom, 2015, pp. 55–60.

[63] E. Contini-Morava. “Noun Classification in Swahili”. In: *African Linguistics at the Crossroads: Papers from Kwaluseni*. Ed. by R. K. Herbert. Rüdiger Köppe Verlag, 1997. Chap. 6.

[64] A. T. Cope. “The grammatical structure of Zulu”. In: *African Studies* 16.4 (1957), pp. 210–220.

[65] N. Dai, J. Liang, X. Qiu, and X. Huang. “Style Transformer: Unpaired Text Style Transfer without Disentangled Latent Representation”. In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Ed. by A. Korhonen, D. R. Traum, and L. Màrquez. Association for Computational Linguistics, 2019, pp. 5997–6007.

[66] R. Dale. “NLP commercialisation in the last 25 years”. In: *Natural Language Engineering* 25.3 (2019), pp. 419–426.

[67] R. Dale. “Natural language generation: The commercial state of the art in 2020”. In: *Natural Language Engineering* 26.4 (2020), pp. 481–487.

[68] R. Dale and E. Reiter. *Building Natural Language Generation Systems*. Cambridge University Press, 2000.

[69] D. Dannélls. “Multilingual text generation from structured formal representations.” PhD thesis. Göteborg, Sweden: Department of Swedish, University of Gothenburg, 2012.

[70] D. Dannélls. "On generating coherent multilingual descriptions of museum objects from Semantic Web ontologies". In: *Proceedings of the Seventh International Natural Language Generation Conference, INLG 2012, 30 May 2012 - 1 June 2012, Starved Rock State Park, Utica, IL, USA*. Ed. by B. D. Eugenio, S. McRoy, A. Gatt, A. Belz, A. Koller, and K. Striegnitz. Association for Computer Linguistics, 2012, pp. 76–84.

[71] D. Dannélls, M. Damova, R. Enache, and M. Chechev. "Multilingual online generation from semantic web ontologies". In: *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*. Ed. by A. Mille, F. Gandon, J. Misselis, M. Rabinovich, and S. Staab. ACM, 2012, pp. 239–242.

[72] B. Davis, R. Enache, J. van Grondelle, and L. Pretorius. "Multilingual Verbalisation of Modular Ontologies Using GF and lemon". In: *Proceedings of the Third International Workshop on Controlled Natural Language, CNL 2012, Zurich, Switzerland, August 29-31, 2012.* Ed. by T. Kuhn and N. E. Fuchs. Vol. 7427. Lecture Notes in Computer Science. Springer, 2012, pp. 167–184.

[73] B. Davis, A. A. Iqbal, A. Funk, V. Tablan, K. Bontcheva, H. Cunningham, and S. Handschuh. "RoundTrip Ontology Authoring". In: *Proceedings on the 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008.* Ed. by A. P. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. W. Finin, and K. Thirunarayan. Vol. 5318. Lecture Notes in Computer Science. Springer, 2008, pp. 50–65.

[74] R. de Oliveira and S. Sripada. "Adapting SimpleNLG for Brazilian Portuguese Realisation". In: *Proceedings of the Eighth International Conference on Natural Language Generation, INLG 2014, 19-21 June 2014, Philadelphia, PA, USA*. Association for Computational Linguistics, 2014, pp. 93–94.

[75] K. Demuth. "Bantu noun class systems: loanword and acquisition evidence of semantic productivity". In: *Systems of Nominal Classification*. Ed. by G. Senft. Cambridge University Press, 2000. Chap. 8, pp. 270–292.

[76] J. P. Denny and C. A. Creider. "The semantics of noun classes in Proto-Bantu". In: *Noun classes and categorization*. Ed. by C. G. Craig. John Benjamins Publishing Company, 1986. Chap. 3, pp. 217–239.

[77] D. Derbyshire and D. L. Payne. "Noun classification systems of Amazonian languages". In: *Amazonian linguistics. Studies in lowland South American languages* (1990), pp. 243–272.

[78] C. M. Doke. *Textbook of Zulu grammar*. 2rd. Longmans Southern Africa, 1931.

[79] P. Dongilli and E. Franconi. "An Intelligent Query Interface with Natural Language Support". In: *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference, Melbourne Beach, Florida, USA, May 11-13, 2006*. Ed. by G. Sutcliffe and R. Goebel. AAAI Press, 2006, pp. 658–663.

[80] W. Du and A. W. Black. “Learning to Order Graph Elements with Application to Multilingual Surface Realization”. In: *Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR 2019)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 18–24.

[81] J. Du Plessis. “The Structure of Nominal Modifiers in Xhosa”. In: *South African Journal of African Languages* 5.2 (1985), pp. 35–42.

[82] J. A. Du Plessis and M. Visser. *Xhosa Syntax*. Via Africa Limited, 1992.

[83] O. Dušek. “Novel Methods for Natural Language Generation in Spoken Dialogue Systems”. PhD thesis. Institute of Formal and Applied Linguistics, Charles University, Prague, 2017.

[84] O. Dušek, J. Novikova, and V. Rieser. “Evaluating the state-of-the-art of End-to-End Natural Language Generation: The E2E NLG challenge”. In: *Computer Speech & Language* 59 (2020), pp. 123–156.

[85] H. Elder. “ADAPT at SR’20: How Preprocessing and Data Augmentation Help to Improve Surface Realization”. In: *Proceedings of the Third Workshop on Multilingual Surface Realisation*. Barcelona, Spain (Online): Association for Computational Linguistics, Dec. 2020, pp. 30–34.

[86] M. Elhadad and J. Robin. “An Overview of SURGE: a Reusable Comprehensive Syntactic Realization Component”. In: *Eighth International Natural Language Generation Workshop, INLG 1996, Herstmonceux Castle, Sussex, UK, June 12-15, 1996 - Posters and Demonstrations*. Ed. by D. Scott, J. Bateman, G. Lapalme, D. McDonald, C. Paris, and K. Linden. 1996, pp. 1–4.

[87] A. Fan and C. Gardent. “Multilingual AMR-to-Text Generation”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*. Ed. by B. Webber, T. Cohn, Y. He, and Y. Liu. Association for Computational Linguistics, 2020, pp. 2889–2901.

[88] F. Farahnak, L. Rafiee, L. Kosseim, and T. Fevens. “The Concordia NLG Surface Realizer at SRST 2019”. In: *Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR 2019)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 63–67.

[89] F. Farahnak, L. Rafiee, L. Kosseim, and T. Fevens. “Surface Realization Using Pretrained Language Models”. In: *Proceedings of the Third Workshop on Multilingual Surface Realisation*. Barcelona, Spain (Online): Association for Computational Linguistics, Dec. 2020, pp. 57–63.

[90] A. C. Fuentes, A. Ramos-Soto, and A. J. B. Diz. “Adapting SimpleNLG to Galician language”. In: *Proceedings of the 11th International Conference on Natural Language Generation, Tilburg University, The Netherlands, November 5-8, 2018*. Ed. by E. Krahmer, A. Gatt, and M. Goudbeek. Association for Computational Linguistics, 2018, pp. 67–72.

[91] I. Funke, R. Helaoui, and A. Härmä. “Interactive health insight miner: an adaptive, semantic-based approach”. In: *Proceedings of the 11th International Conference on Natural Language Generation, Tilburg University, The Netherlands, November 5-8, 2018*. Ed. by E. Krahmer, A. Gatt, and M. Goudbeek. Association for Computational Linguistics, 2018, pp. 478–479.

[92] C. Gardent, A. Shimorina, S. Narayan, and L. Perez-Beltrachini. “The WebNLG Challenge: Generating Text from RDF Data”. In: *Proceedings of the 10th International Conference on Natural Language Generation, INLG 2017, Santiago de Compostela, Spain, September 4-7, 2017*. Ed. by J. M. Alonso, A. Bugarín, and E. Reiter. Association for Computational Linguistics, 2017, pp. 124–133.

[93] A. Gatt and E. Krahmer. “Survey of the State of the Art in Natural Language Generation: Core Tasks, Applications and Evaluation”. In: *Journal of Artificial Intelligence Research* 61 (2018), pp. 65–170.

[94] A. Gatt and E. Reiter. “SimpleNLG: A Realisation Engine for Practical Applications”. In: ed. by E. Krahmer and M. Theune. March 30-31Athens, Greece. Association for Computer Linguistics, 2009, pp. 90–93.

[95] F. Gillis-Webber, S. Tittel, and C. M. Keet. “A Model for Language Annotations on the Web”. In: *Knowledge Graphs and Semantic Web*. Ed. by B. Villazón-Terrazas and Y. Hidalgo-Delgado. Cham: Springer International Publishing, 2019, pp. 1–16.

[96] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang. “HermiT: An OWL 2 Reasoner”. In: *Journal of Automated Reasoning* 53.3 (Oct. 2014), pp. 245–269.

[97] L. A. Goodman. “Snowball Sampling”. In: *The Annals of Mathematical Statistics* 32.1 (1961), pp. 148–170.

[98] J. C. van Grondelle and M. Gülpers. “Specifying Flexible Business Processes Using Pre and Post Conditions”. In: *Proceedings of the 4th IFIP WG 8.1 Working Conference on The Practice of Enterprise Modeling, PoEM 2011*. Ed. by P. Johannesson, J. Krogstie, and A. L. Opdahl. Vol. 92. Lecture Notes in Business Information Processing. November 2-3, Oslo, Norway. Springer, 2011, pp. 38–51.

[99] A. S. Grover, G. B. Van Huyssteen, and M. W. Pretorius. “The South African human language technology audit”. In: *Language resources and evaluation* 45.3 (2011), pp. 271–288.

[100] N. Gruzitis, G. Nespore, and B. Saulite. “Verbalizing Ontologies in Controlled Baltic Languages”. In: *Human Language Technologies - The Baltic Perspective - Proceedings of the Fourth International Conference Baltic HLT 2010, Riga, Latvia, October 7-8, 2010*. Ed. by I. Skadina and A. Vasiljevs. Vol. 219. Frontiers in Artificial Intelligence and Applications. IOS Press, 2010, pp. 187–194.

[101] N. Gruzitis, G. Nespore, and B. Saulite. “Verbalizing Ontologies in Controlled Baltic Languages”. In: *CoRR* (2012). arXiv: `1211.0418`.

[102] A. Gubbala, A. H. Ladiwala, U. Naik, and P. Cornelius. *Citizen Friendly Report of diversitydatakids.org*. Tech. rep. San Josè State University, 2021.

[103] Y. Guo, D. Hogan, and J. van Genabith. "DCU at Generation Challenges 2011 Surface Realisation Track". In: *Proceedings of the 13th European Workshop on Natural Language Generation*. Nancy, France: Association for Computational Linguistics, Sept. 2011, pp. 227–229.

[104] T. A. Halpin and M. Curland. "Automated Verbalization for ORM 2". In: *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, OTM Confederated International Workshops and Posters, AWeSOMe, CAMS, COMINF, IS, KSinBIT, MIOS-CIAO, MONET, OnToContent, ORM, PerSys, OTM Academy Doctoral Consortium, RDDS, SWWS, and SeBGIS 2006, Montpellier, France, October 29 - November 3, 2006. Proceedings, Part II*. Ed. by R. Meersman, Z. Tari, and P. Herrero. Vol. 4278. Lecture Notes in Computer Science. Springer, 2006, pp. 1181–1190.

[105] T. A. Halpin and J. P. Wijbenga. "FORML 2". In: *Proccedings of the 11th International Workshop on Enterprise, Business-Process and Information Systems Modeling, BPMDS 2010, and Proceedings of the 15th International Conference, EMMSAD 2010, held at CAiSE 2010, Hammamet, Tunisia, June 7-8, 2010*. Ed. by I. Bider, T. A. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, and R. Ukor. Vol. 50. Lecture Notes in Business Information Processing. Springer, 2010, pp. 247–260.

[106] V. Harrison and M. A. Walker. "Neural Generation of Diverse Questions using Answer Focus, Contextual and Linguistic Features". In: *Proceedings of the 11th International Conference on Natural Language Generation, Tilburg University, The Netherlands, November 5-8, 2018*. Ed. by E. Krahmer, A. Gatt, and M. Goudbeek. Association for Computational Linguistics, 2018, pp. 296–306.

[107] R. K. Herbert. "Labial palatalization in Nguni and Sotho languages: internal and external evidence". In: *South African Journal of African Languages* 10.2 (1990), pp. 74–80.

[108] D. Hewlett, A. Kalyanpur, V. Kolovski, and C. Halaschek-Wiener. "Proc. of the Workshop on End-User Semantic Web Interaction, held in conjunction with the 4th Internatioanl Semantic Web conference (ISWC)". In: *Proc. of the Workshop on End-User Semantic Web Interaction, held in conjunction with the 4th Internatioanl Semantic Web conference (ISWC)*. Galway, Ireland, November 7, 2005. CEUR-WS.org, 2005.

[109] F. Hielkema, C. Mellish, and P. Edwards. "Using WYSIWYM to Create an Open-ended Interface for the Semantic Grid". In: *Proceedings of the Eleventh European Workshop on Natural Language Generation, ENLG 2007, Schloss Dagstuhl, Germany, June 17-20, 2007*. Ed. by S. Busemann. 2007.

[110] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. "The Curious Case of Neural Text Degeneration". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[111] X. Hong, E. Chang, and V. Demberg. "Improving Language Generation from Feature-Rich Tree-Structured Data with Relational Graph Convolutional Encoders". In: *Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR 2019)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 75–80.

[112] B. A. Hossain, G. Rajan, and R. Schwitter. "CNL-ER: A Controlled Natural Language for Specifying and Verbalising Entity Relationship Models". In: *Proceedings of the 17th Annual Workshop of the Australasian Language Technology Association, ALTA 2019*. Ed. by M. Mistica, M. Piccardi, and A. MacKinlay. December 4-6, Sydney, Australia. Australasian Language Technology Association, 2019, pp. 126–135.

[113] E. Hovy, G. van Noord, G. Neumann, and J. Bateman. "Language generation". In: *Survey of the State of the Art in Human Language Technology*. Ed. by R. Cole, J. Mariani, H. Uszkoreit, G. B. Varile, A. Zaenen, A. Zampolli, and V. Zue. Springer Berlin Heidelberg, 1997. Chap. 4, pp. 139–163.

[114] E. H. Hovy. "Planning Coherent Multisentential Text". In: *Proceedings of 26th Annual Meeting of the Association for Computational Linguistics, 7-10 June 1988, State Univerity of New York at Buffalo, Buffalo, New York, USA*. Ed. by J. R. Hobbs. ACL, 1988, pp. 163–169.

[115] B. Howald, R. Kondadadi, and F. Schilder. "Domain Adaptable Semantic Clustering in Statistical NLG". In: *Proceedings of the 10th International Conference on Computational Semantics, March 19-22, 2013, University of Potsdam, Germany*. ACL, 2013, pp. 143–154.

[116] M. Ihori, R. Masumura, N. Makishima, T. Tanaka, A. Takashima, and S. Orihashi. "Memory Attentive Fusion: External Language Model Integration for Transformer-based Sequence-to-Sequence Model". In: *Proceedings of the 13th International Conference on Natural Language Generation, INLG 2020, Dublin, Ireland, December 15-18, 2020*. Ed. by B. Davis, Y. Graham, J. D. Kelleher, and Y. Sripada. Association for Computational Linguistics, 2020, pp. 1–6.

[117] A. Isard and J. Knox. "Automatic Generation of Student Report Cards". In: *Proceedings of the Ninth International Natural Language Generation Conference, INLG 2016, September 5-8, 2016, Edinburgh, UK*. Ed. by A. Isard, V. Rieser, and D. Gkatzia. Association for Computer Linguistics, 2016, pp. 207–211.

[118] M. Jarrar, C. M. Keet, and P. Dongilli. *Multilingual verbalization of ORM conceptual models and axiomatized ontologies*. Tech. rep. Belgium: Starlab, Vrije Universiteit Brussel, Feb. 2006.

[119] J. Jawitz. *The challenge of teaching large classes in higher education in South Africa : a battle to be waged outside the classroom*. Ed. by D. J. Hornsby, R. Osman, and J. De Matos-ala. Cape Town: University of Cape Town, 2013.

[120] R. de Jong and M. Theune. "Going Dutch: Creating SimpleNLG-NL". In: *Proceedings of the 11th International Conference on Natural Language Generation, Tilburg University, The Netherlands, November 5-8, 2018*. Ed. by E. Krahmer, A. Gatt, and M. Goudbeek. Association for Computational Linguistics, 2018, pp. 73–78.

[121] J. Juraska and M. A. Walker. "Attention Is Indeed All You Need: Semantically Attention-Guided Decoding for Data-to-Text NLG". In: *Proceedings of the 14th International Conference on Natural Language Generation, INLG 2021, Aberdeen, Scotland, UK, 20-24 September, 2021*. Ed. by A. Belz, A. Fan, E. Reiter, and Y. Sripada. Association for Computational Linguistics, 2021, pp. 416–431.

[122] S. Kahane. "The Meaning-Text Theory". In: vol. 1. 25. De Gruyter, 2003, pp. 546–570.

[123] M. Kale and A. Rastogi. "Template Guided Text Generation for Task-Oriented Dialogue". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*. Ed. by B. Webber, T. Cohn, Y. He, and Y. Liu. Association for Computational Linguistics, 2020, pp. 6505–6520.

[124] K. Kaljurand and N. E. Fuchs. "Verbalizing OWL in Attempto Controlled English". In: *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions, Innsbruck, Austria, June 6-7, 2007*. Ed. by C. Golbreich, A. Kalyanpur, and B. Parsia. Vol. 258. CEUR Workshop Proceedings. CEUR-WS.org, 2007.

[125] D. Kalpakchi and J. Boye. "BERT-based distractor generation for Swedish reading comprehension questions using a small-scale dataset". In: *Proceedings of the 14th International Conference on Natural Language Generation, INLG 2021, Aberdeen, Scotland, UK, 20-24 September, 2021*. Ed. by A. Belz, A. Fan, E. Reiter, and Y. Sripada. Association for Computational Linguistics, 2021, pp. 387–403.

[126] Z. Kasner, S. Mille, and O. Dusek. "Text-in-Context: Token-Level Error Detection for Table-to-Text Generation". In: *Proceedings of the 14th International Conference on Natural Language Generation, INLG 2021, Aberdeen, Scotland, UK, 20-24 September, 2021*. Ed. by A. Belz, A. Fan, E. Reiter, and Y. Sripada. Association for Computational Linguistics, 2021, pp. 259–265.

[127] R. T. Kasper. "A Flexible Interface for Linking Applications to Penman's Sentence Generator". In: *Speech and Natural Language: Proceedings of a Workshop Held at Philadelphia, USA, HLT 1989, February 21-23, 1989*. ACL, 1989.

[128] F. Katamba. "Bantu Nominal Morphology". In: *The Bantu Languages*. Ed. by D. Nurse and G. Philippson. Routledge, 2014. Chap. 7, pp. 103–120.

[129] C. M. Keet. “An assessment of orthographic similarity measures for several African languages”. In: *CoRR* abs/1608.03065 (2016).

[130] C. M. Keet. *An Introduction to Ontology Engineering*. University of Cape Town, July 2018.

[131] C. M. Keet. “The African wildlife ontology tutorial ontologies”. In: *Journal of Biomedical Semantics* 11.1 (2020), p. 4.

[132] C. M. Keet and A. Artale. “Representing and reasoning over a taxonomy of part-whole relations”. In: *Appl. Ontology* 3.1-2 (2008), pp. 91–110.

[133] C. M. Keet and L. Khumalo. “Toward Verbalizing Ontologies in isiZulu”. In: *Proc. of CNL’014*. Ed. by B. Davis, K. Kaljurand, and T. Kuhn. Vol. 8625. LNCS. August 20-22, Galway, Ireland. Springer, 2014, pp. 78–89.

[134] C. M. Keet and L. Khumalo. “On the verbalization patterns of part-whole relations in isiZulu”. In: *Proceedings of the Ninth International Natural Language Generation Conference, INLG 2016, September 5-8, 2016, Edinburgh, UK*. Ed. by A. Isard, V. Rieser, and D. Gkatzia. Association for Computer Linguistics, 2016, pp. 174–183.

[135] C. M. Keet and L. Khumalo. “Grammar Rules for the isiZulu Complex Verb”. In: *Southern African Linguistics and Applied Language Studies* 35.2 (2017), pp. 183–200.

[136] C. M. Keet and L. Khumalo. “Toward a Knowledge-to-Text Controlled Natural Language of isiZulu”. In: *Language Resources and Evaluation* 51.1 (2017), pp. 131–157.

[137] C. M. Keet, Z. Mahlaza, and M.-J. Antia. “CLaRO: A Controlled Language for Authoring Competency Questions”. In: *Garoufallou E., Fallucchi F., William De Luca E. (eds) Metadata and Semantic Research. MTSR2019. Communications in Computer and Information Science, vol 1057*. Springer, Cham., 2019, pp. 3–15.

[138] C. M. Keet, M. Xakaza, and L. Khumalo. “Verbalising OWL ontologies in isiZulu with Python”. In: *The Semantic Web: ESWC 2017 Satellite Events*. Ed. by E. Blomqvist, K. Hose, H. Paulheim, A. Lawrynowicz, F. Ciravegna, and O. Hartig. Vol. 10577. LNCS. 30 May - 1 June 2017, Portoroz, Slovenia. Springer, 2017, pp. 59–64.

[139] R. Kittredge, A. Polguère, and E. Goldberg. “Synthesizing Weather Forecasts from Formated Data”. In: *Proceedings of the 11th Coference on Computational Linguistics*. COLING ’86. Bonn, Germany: Association for Computational Linguistics, 1986, pp. 563–565.

[140] K. Knight and V. Hatzivassiloglou. “Two-level, Many-Paths Generation”. In: *CoRR* abs/cmp-lg/9506010 (1995). arXiv: `cmp-lg/9506010`.

[141] J. Knox. “Playing with student data: The learning analytics report card (LARC)”. In: *Proceedings of the of the 7th International Learning Analytics & Knowledge Conference (LAK17): Practitioner Track*. Vancouver, Canada: The Society for Learning Analytics Research, Mar. 2017, pp. 43–49.

[142] P. Koehn, F. J. Och, and D. Marcu. “Statistical Phrase-Based Translation”. In: *Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, HLT-NAACL 2003, Edmonton, Canada, May 27 - June 1, 2003*. Association of Computational Linguistics, 2003.

[143] R. Kondadadi, B. Howald, and F. Schilder. “A Statistical NLG Framework for Aggregated Planning and Realization”. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, 4-9 August 2013, Sofia, Bulgaria*. ACL, 2013, pp. 1406–1415.

[144] R. Kondadadi, B. Howald, and F. Schilder. “A Statistical NLG Framework for Aggregated Planning and Realization”. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 1: Long Papers*. ACL, 2013, pp. 1406–1415.

[145] Á. Kovács, E. Ács, J. Ács, A. Kornai, and G. Recski. “BME-UW at SRST-2019: Surface realization with Interpreted Regular Tree Grammars”. In: *Proceedings of the 2nd Workshop on Multilingual Surface Realisation, MSR@EMNLP-IJCNLP 2019, Hong Kong, China, November 3, 2019*. Ed. by S. Mille, A. Belz, B. Bohnet, Y. Graham, and L. Wanner. Association for Computational Linguistics, 2019, pp. 35–40.

[146] Z. Kuanzhuo, L. Lin, and W. Zhao. “SimpleNLG-TI: Adapting SimpleNLG to Tibetan”. In: *Proceedings of the 13th International Conference on Natural Language Generation, INLG 2020, Dublin, Ireland, December 15-18, 2020*. Association for Computational Linguistics, 2020, pp. 86–90.

[147] K. Kuratowski. *Introduction to set theory and topology*. Pergamon Press, 2014.

[148] I. Langkilde. “Forest-Based Statistical Sentence Generation”. In: *6th Applied Natural Language Processing Conference, ANLP 2000, Seattle, Washington, USA, April 29 - May 4, 2000*. Association of Computational Linguistics, 2000, pp. 170–177.

[149] I. Langkilde-Geary and K. Knight. “HALogen Statistical Sentence Generator”. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (Demonstrations Session), July 6-12, 2002, Philadelphia, PA, USA*. 2002, pp. 102–103.

[150] G. Lapalme. “The jsRealB Text Realizer: Organization and Use Cases”. In: *CoRR* (2020). arXiv: 2012.15425.

[151] F. Lareau, F. Lambrey, I. Dubinskaite, D. Galarreta-Piquette, and M. Nejat. "GenDR: A Generic Deep Realizer with Complex Lexicalization". In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018*. European Language Resources Association (ELRA), 2018.

[152] B. Lavoie, R. I. Kittredge, T. Korelsky, and O. Rambow. "A Framework for MT and Multilingual NLG Systems Based on Uniform Lexico-Structural Processing". In: *6th Applied Natural Language Processing Conference, ANLP 2000, Seattle, Washington, USA, April 29 - May 4, 2000*. ACL, 2000, pp. 60–67.

[153] B. Lavoie and O. Rainbow. "A Fast and Portable Realizer for Text Generation Systems". In: *5th Applied Natural Language Processing Conference, ANLP 1997, Marriott Hotel, Washington, USA, March 31 - April 3, 1997*. ACL, 1997, pp. 265–268.

[154] R. Lebret, D. Grangier, and M. Auli. "Neural Text Generation from Structured Data with Application to the Biography Domain". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*. Ed. by J. Su, X. Carreras, and K. Duh. The Association for Computational Linguistics, 2016, pp. 1203–1213.

[155] J. Lee. "Stable Style Transformer: Delete and Generate Approach with Encoder-Decoder for Text Style Transfer". In: *Proceedings of the 13th International Conference on Natural Language Generation, INLG 2020, Dublin, Ireland, December 15-18, 2020*. Ed. by B. Davis, Y. Graham, J. D. Kelleher, and Y. Sripada. Association for Computational Linguistics, 2020, pp. 195–204.

[156] Y. Leng, F. Portet, C. Labbé, and R. Qader. "Controllable Neural Natural Language Generation: comparison of state-of-the-art control strategies". In: *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*. Dublin, Ireland (Virtual): Association for Computational Linguistics, Dec. 2020, pp. 34–39.

[157] F. Liang, R. Stevens, and A. L. Rector. "OntoVerbal-M: a Multilingual Verbaliser for SNOMED CT". In: *Proceedings of the 2nd International Workshop on the Multilingual Semantic Web, Bonn, Germany, October 23, 2011*. Ed. by E. Montiel-Ponsoda, J. P. McCrae, P. Buitelaar, and P. Cimiano. Vol. 775. CEUR Workshop Proceedings. CEUR-WS.org, 2011, pp. 13–24.

[158] S. F. Liang, D. Scott, R. Stevens, and A. L. Rector. "OntoVerbal: a Generic Tool and Practical Application to SNOMED CT". In: *CoRR* abs/1312.2798 (2013). arXiv: 1312.2798.

[159] S. F. Liang, R. Stevens, D. Scott, and A. L. Rector. "Automatic Verbalisation of SNOMED Classes Using OntoVerbal". In: *Artificial Intelligence in Medicine - 13th Conference on Artificial Intelligence in Medicine, AIME 2011, Bled, Slovenia, July 2-6, 2011. Proceedings*. Ed. by M. Peleg, N. Lavrac, and C. Combi. Vol. 6747. Lecture Notes in Computer Science. Springer, 2011, pp. 338–342.

[160] S. H. Lim and T. A. Halpin. “Automated Verbalization of ORM Models in Malay and Mandarin”. In: *International Journal of Information System Modeling and Design* 7.4 (2016), pp. 1–16.

[161] Y. Liu and M. Lapata. “Text Summarization with Pretrained Encoders”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Ed. by K. Inui, J. Jiang, V. Ng, and X. Wan. Association for Computational Linguistics, 2019, pp. 3728–3738.

[162] Y. Lu, Z. Li, D. He, Z. Sun, B. Dong, T. Qin, L. Wang, and T. Liu. “Understanding and Improving Transformer From a Multi-Particle Dynamic System Point of View”. In: *CoRR* abs/1906.02762 (2019). arXiv: 1906.02762.

[163] O. Lyudovyk and C. Weng. “SNOMEDtxt: Natural Language Generation from SNOMED Ontology”. In: *MEDINFO 2019: Health and Wellbeing e-Networks for All - Proceedings of the 17th World Congress on Medical and Health Informatics, Lyon, France, 25-30 August 2019*. Ed. by L. Ohno-Machado and B. Séroussi. Vol. 264. Studies in Health Technology and Informatics. IOS Press, 2019, pp. 1263–1267.

[164] A. Madsack, J. Heininger, N. Davaasambuu, V. Voronik, M. Käufl, and R. Weißgraeber. “AX Semantics’ Submission to the Surface Realization Shared Task 2018”. In: *Proceedings of the First Workshop on Multilingual Surface Realisation*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 54–57.

[165] S. Mahamood and M. Zembrzuski. “Hotel Scribe: Generating High Variation Hotel Descriptions”. In: *Proceedings of the 12th International Conference on Natural Language Generation, INLG 2019, Tokyo, Japan, October 29 - November 1, 2019*. 2019, pp. 391–396.

[166] Z. Mahlaza and C. M. Keet. “Formalisation and classification of grammar and template-mediated techniques to model and ontology verbalisation”. In: *International Journal of Metadata, Semantics and Ontologies* 14.3 (2020), pp. 249–262.

[167] Z. Mahlaza and C. M. Keet. “Measuring Verb Similarity Using Binary Coefficients with Application to isiXhosa and isiZulu”. In: *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists*. SAICSIT ’18. Port Elizabeth, South Africa: ACM, 2018, pp. 65–71.

[168] Z. Mahlaza and C. M. Keet. “A classification of grammar-infused templates for ontology and model verbalisation”. In: *Metadata and Semantic Research - 13th International Conference, MTSR 2019, Rome, Italy, October 28-31, 2019*. 2019.

[169] Z. Mahlaza and C. M. Keet. “OWLSIZ: An isiZulu CNL for structured knowledge validation”. In: *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*. Dublin, Ireland (Virtual): Association for Computational Linguistics, Dec. 2020, pp. 15–25.

[170] Z. Mahlaza and C. M. Keet. “Surface Realisation Architecture for Low-Resourced African Languages”. In: *ACM Trans. Asian Low-Resour. Lang. Inf. Process.* (Oct. 2022). Just Accepted.

[171] Z. Mahlaza, C. M. Keet, J. Dunn, and M. Poulter. “An evaluation of template and ML-based generation of user-readable text from a knowledge graph”. In: *CoRR* abs/2106.14613 (2021). arXiv: 2106.14613.

[172] J. Maho. *A Comparative Study of Bantu Noun Classes*. Acta Universitatis Gothoburgunsis, 1999.

[173] J. Maho. *NUGL OnlineThe online version of the New Updated Guthrie List, a referential classification of the Bantu languages*. https://brill.com/fileasset/downloads_products/35125_Bantu-New-updated-Guthrie-List.pdf. Accessed: 25 August 2021. 2009.

[174] J. Maho. “A classification of the Bantu Languages: An Update of Guthrie’s referential system”. In: *The Bantu Languages*. Ed. by D. Nurse and G. Philippson. Routledge, 2014. Chap. 7, pp. 639–651.

[175] F. Mairesse, M. Gasic, F. Jurcícek, S. Keizer, B. Thomson, K. Yu, and S. J. Young. “Phrase-Based Statistical Language Generation Using Graphical Models and Active Learning”. In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, ACL 2010, July 11-16, 2010, Uppsala, Sweden*. Ed. by J. Hajic, S. Carberry, and S. Clark. Association for Computer Linguistics, 2010, pp. 1552–1561.

[176] J. H. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond. “The Scratch Programming Language and Environment”. In: *ACM Transactions on Computing Education* 10.4 (2010), 16:1–16:15.

[177] W. C. Mann and C. M. Matthiessen. *Nigel: A Systemic Grammar for Text Generation.* Tech. rep. ISI/RR-83-105. Information Sciences Institute, Univeristy of Southern California, Marina Del Rey, USA, 1983.

[178] W. C. Mann. “An Overview of the Nigel Text Generation Grammar”. In: *21st Annual Meeting of the Association for Computational Linguistics, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, June 15-17, 1983*. Ed. by M. P. Marcus. ACL, 1983, pp. 79–84.

[179] W. C. Mann. “An Overview of the Penman Text Generation System”. In: *Proceedings of the National Conference on Artificial Intelligence, Washington, D.C., USA, August 22-26, 1983*. Ed. by M. R. Genesereth. AAAI Press, 1983, pp. 261–265.

[180] W. C. Mann, M. Bates, B. Grosz, D. D. McDonald, K. R. Mckeown, and W. Swartout. "Text Generation". In: *American Journal of Computational Linguistics* 8.2 (1982), pp. 62–69.

[181] L. Marais. "Approximating a Zulu GF concrete syntax with a neural network for natural language understanding". In: *Proc. of CNL 2020/21*. Amsterdam, Netherlands: Association for Computational Linguistics, Sept. 2021.

[182] T. Marciniak and M. Strube. "Classification-Based Generation Using TAG". In: *Proceedings on the Third International Conference on Natural Language Generation, INLG 2004, Brockenhurst, UK, July 14-16, 2004*. Ed. by A. Belz, R. Evans, and P. Piwek. Vol. 3123. Lecture Notes in Computer Science. Springer, 2004, pp. 100–109.

[183] L. Martinus and J. Z. Abbott. "A Focus on Neural Machine Translation for African Languages". In: *arXiv e-prints* (June 2019). arXiv: `1906.05685 [cs.CL]`.

[184] J. May and J. Priyadarshi. "SemEval-2017 Task 9: Abstract Meaning Representation Parsing and Generation". In: *Proceedings of the 11th International Workshop on Semantic Evaluation, SemEval@ACL 2017, Vancouver, Canada, August 3-4, 2017*. Ed. by S. Bethard, M. Carpuat, M. Apidianaki, S. M. Mohammad, D. M. Cer, and D. Jurgens. Association for Computational Linguistics, 2017, pp. 536–545.

[185] A. Mazzei and V. Basile. "The DipInfoUniTo Realizer at SRST'19: Learning to Rank and Deep Morphology Prediction for Multilingual Surface Realization". In: *Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR 2019)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 81–87.

[186] A. Mazzei, C. Battaglino, and C. Bosco. "SimpleNLG-IT: Adapting SimpleNLG to Italian". In: *Proceedings of the Ninth International Conference on Natural Language Generation, INLG 2016, September 5-8, 2016, Edinburgh, UK*. Association for Computational Linguistics, 2016, pp. 184–192.

[187] J. McCrae, G. Aguado-de-Cea, P. Buitelaar, P. Cimiano, T. Declerck, A. Gómez Pérez, J. Gracia, L. Hollink, E. Montiel-Ponsoda, D. Spohr, et al. *The lemon cookbook*. Tech. rep. Monnet Project, June 2012.

[188] K. McKeown. *Text generation*. Cambridge University Press, 1992.

[189] J. McLaren. *A Xhosa Grammar, Revised and Re-Written in the New Orthography, Edited by G. H. Welsh*. Longmans, Green and Company, 1944.

[190] S. W. McRoy, S. Channarukul, and S. S. Ali. "YAG: A Template-based Generator for Real-time Systems". In: *Proceedings of the First International Conference on Natural Language Generation - Volume 14*. INLG '00. Mitzpe Ramon, Israel: Association for Computational Linguistics, 2000, pp. 264–267.

[191] S. W. McRoy, S. Channarukul, and S. S. Ali. “An augmented template-based approach to text realization”. In: *Natural Language Engineering* 9.4 (2003), pp. 381–420.

[192] A. E. Meeussen. “Bantu Grammatical Reconstructions”. In: *Africana linguistica* 3.1 (1967), pp. 79–121.

[193] I. Melcuk. *Dependency Syntax: Theory and Practice*. SUNY press, 1988.

[194] C. Mellish and J. Z. Pan. “Finding Subsumers for Natural Language Presentation”. In: *Proceedings of the 2006 International Workshop on Description Logics (DL2006), Windermere, Lake District, UK, May 30 - June 1, 2006*. Ed. by B. Parsia, U. Sattler, and D. Toman. Vol. 189. CEUR Workshop Proceedings. CEUR-WS.org, 2006.

[195] C. Mellish and J. Z. Pan. “Natural language directed inference from ontologies”. In: *Artif. Intell.* 172.10 (2008), pp. 1285–1315.

[196] C. Mellish, D. Scott, L. J. Cahill, D. S. Paiva, R. Evans, and M. Reape. “A Reference Architecture for Natural Language Generation Systems”. In: *Nat. Lang. Eng.* 12.1 (2006), pp. 1–34.

[197] S. Mille, A. Belz, B. Bohnet, Y. Graham, E. Pitler, and L. Wanner. “The First Multilingual Surface Realisation Shared Task (SR’18): Overview and Evaluation Results”. In: *Proceedings of the First Workshop on Multilingual Surface Realisation*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 1–12.

[198] S. Mille, A. Belz, B. Bohnet, Y. Graham, and L. Wanner. “The Second Multilingual Surface Realisation Shared Task (SR’19): Overview and Evaluation Results”. In: *Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR 2019)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 1–17.

[199] S. Mille, A. Belz, B. Bohnet, T. Castro Ferreira, Y. Graham, and L. Wanner. “The Third Multilingual Surface Realisation Shared Task (SR’20): Overview and Evaluation Results”. In: *Proceedings of the Third Workshop on Multilingual Surface Realisation*. Barcelona, Spain (Online): Association for Computational Linguistics, Dec. 2020, pp. 1–20.

[200] P. Molins and G. Lapalme. “JSrealB: A Bilingual Text Realizer for Web Programming”. In: *ENLG 2015 - Proceedings of the 15th European Workshop on Natural Language Generation, 10-11 September 2015, University of Brighton, Brighton, UK*. Association for Computer Linguistics, 2015, pp. 109–111.

[201] C. Moors, I. Wilken, K. Calteaux, and T. Gumede. “Human Language Technology Audit 2018: Analysing the Development Trends in Resource Availability in All South African Languages”. In: *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists*. SAICSIT ’18. Port Elizabeth, South Africa: ACM, 2018, pp. 296–304.

[202] A. Moryossef, Y. Goldberg, and I. Dagan. “Step-by-Step: Separating Planning from Realization in Neural Data-to-Text Generation”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Ed. by J. Burstein, C. Doran, and T. Solorio. Association for Computational Linguistics, 2019, pp. 2267–2277.

[203] N. Mtsatse and C. Combrinck. “Dialects matter: The influence of dialects and code-switching on the literacy and numeracy achievements of isiXhosa Grade 1 learners in the Western Cape”. In: *Journal of Education* 72 (2018), pp. 21–37.

[204] S. Naidoo. “The palatalisation process in isiZulu revisited”. In: *South African Journal of African Languages* 22.1 (2002), pp. 59–69.

[205] H. Nakanishi, Y. Miyao, and J. Tsujii. “Probabilistic Models for Disambiguation of an HPSG-Based Chart Generator”. In: *Proceedings of the Ninth International Workshop on Parsing Technology, IWPT 2005, Vancouver, Canada, October 9-10, 2005*. Ed. by H. Bunt, R. Malouf, and A. Lavie. Association for Computational Linguistics, 2005, pp. 93–102.

[206] S. Narayan and C. Gardent. *Deep Learning Approaches to Text Production*. Morgan & Claypool Publishers, 2020.

[207] W. Ng’ang’a. “Building Swahili Resource Grammars for the Grammatical Framework”. In: *Shall We Play the Festschrift Game? Essays on the Occasion of Lauri Carlson’s 60th Birthday*. Ed. by D. Santos, K. Lindén, and W. Ng’ang’a. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 215–226.

[208] A.-C. Ngonga Ngomo, D. Moussallem, and L. Bühmann. “A Holistic Natural Language Generation Framework for the Semantic Web”. In: *arXiv preprint arXiv:1911.01248* (2019).

[209] V. Nguyen, T. C. Son, and E. Pontelli. “Natural Language Generation from Ontologies”. In: *Proceedings on the 21th International Symposium on Practical Aspects of Declarative Languages, PADL 2019, Lisbon, Portugal, January 14-15, 2019*. Ed. by J. J. Alferes and M. Johansson. Vol. 11372. Lecture Notes in Computer Science. Springer, 2019, pp. 64–81.

[210] V. S. Nomlomo. “Language variation in the Transkeian Xhosa speech community and its impact on children’s education”. MA thesis. University of Cape Town, 1993.

[211] J. Novikova, O. Dusek, and V. Rieser. “The E2E Dataset: New Challenges For End-to-End Generation”. In: *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue, Saarbrücken, Germany, August 15-17, 2017*. Ed. by K. Jokinen, M. Stede, D. DeVault, and A. Louis. Association for Computational Linguistics, 2017, pp. 201–206.

[212] E. Nyoni and B. A. Bassett. “Low-Resource Neural Machine Translation for Southern African Languages”. In: *arXiv e-prints* (Apr. 2021). arXiv: 2104.00366 [cs.CL].

[213] J. Obeid and E. Hoque. “Chart-to-Text: Generating Natural Language Descriptions for Charts by Adapting the Transformer Model”. In: *Proceedings of the 13th International Conference on Natural Language Generation, INLG 2020, Dublin, Ireland, December 15-18, 2020*. Ed. by B. Davis, Y. Graham, J. D. Kelleher, and Y. Sripada. Association for Computational Linguistics, 2020, pp. 138–147.

[214] J. Oosthuysen. *The Grammar of isiXhosa*. SUN press, 2017.

[215] D. M. Oppenheimer, T. Meyvis, and N. Davidenko. “Instructional manipulation checks: Detecting satisficing to increase statistical power”. In: *Journal of Experimental Social Psychology* 45.4 (2009), pp. 867–872.

[216] S. Packham and H. Suleman. “Crowdsourcing a Text Corpus is not a Game”. In: *Digital Libraries: Providing Quality Information - 17th International Conference on Asia-Pacific Digital Libraries, ICADL 2015, Seoul, Korea, December 9-12, 2015, Proceedings*. Ed. by R. B. Allen, J. Hunter, and M. L. Zeng. Vol. 9469. Lecture Notes in Computer Science. Springer, 2015, pp. 225–234.

[217] G. B. Palmer and C. Woodman. “Ontological classifiers as polycentric categories, as seen in Shona class 3 nouns”. In: *Explorations in Linguistic Relativity*. Ed. by M. Pütz and M. Verspoor. John Benjamins Publishing company, 2000. Chap. 12, pp. 225–249.

[218] K. Papineni, S. Roukos, T. Ward, and W. Zhu. “Bleu: a Method for Automatic Evaluation of Machine Translation”. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*. ACL, 2002, pp. 311–318.

[219] A. P. Parikh, X. Wang, S. Gehrmann, M. Faruqui, B. Dhingra, D. Yang, and D. Das. “ToTTo: A Controlled Table-To-Text Generation Dataset”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*. Ed. by B. Webber, T. Cohn, Y. He, and Y. Liu. Association for Computational Linguistics, 2020, pp. 1173–1186.

[220] L. Perez-Beltrachini and M. Lapata. “Bootstrapping Generators from Noisy Data”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*. Ed. by M. A. Walker, H. Ji, and A. Stent. Association for Computational Linguistics, 2018, pp. 1516–1527.

[221] P. Piwek. “A Flexible Pragmatics-Driven Language Generator for Animated Agents”. In: *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2003, April 12-17, 2003, Agro Hotel, Budapest, Hungary*. Association for Computer Linguistics, 2003, pp. 151–154.

[222] L. Posthumus. *A systemized explanation for vowel phoneme change in the inadmissible phonological structure /VV/ in Zulu*. Working paper. Johannesburg, South Africa: University of Johannesburg, Dec. 2016.

[223] M. Poveda-Villalón, A. Gómez-Pérez, and M. C. Suárez-Figueroa. "Oops!(Ontology Pitfall Scanner!): An on-Line Tool for Ontology Evaluation". In: *International Journal on Semantic Web and Information Systems (IJSWIS)* 10.2 (2014), pp. 7–34.

[224] R. Power. "Towards a Generation-Based Semantic Web Authoring Tool". In: *Proceedings of the 12th European Workshop on Natural Language Generation, ENLG 2009, March 30-31, 2009, Athens, Greece*. Ed. by E. Krahmer and M. Theune. The Association for Computer Linguistics, 2009, pp. 9–15.

[225] R. Power. "OWL Simplified English: A Finite-State Language for Ontology Editing". In: *Controlled Natural Language - Third International Workshop, CNL 2012, Zurich, Switzerland, August 29-31, 2012. Proceedings*. Lecture Notes in Artificial Intelligence. Springer, 2012, pp. 44–60.

[226] R. Power and A. Third. "Expressing OWL axioms by English sentences: dubious in theory, feasible in practice". In: *Proceedings of the 23rd International Conference on Computational Linguistics, COLING 2010, Posters Volume, 23-27 August 2010, Beijing, China*. Ed. by C. Huang and D. Jurafsky. Chinese Information Processing Society of China, 2010, pp. 1006–1013.

[227] L. Pretorius and S. Bosch. "Finite State Morphology of the Nguni Language Cluster: Modelling and Implementation Issues". In: *Finite-State Methods and Natural Language Processing*. Ed. by A. Yli-Jyrä, A. Kornai, J. Sakarovitch, and B. Watson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 123–130.

[228] L. Pretorius and S. Bosch. *ZulMorph*. `https://portal.sadilar.org/FiniteState/`. 2019.

[229] L. Pretorius, L. Marais, and A. Berg. "A GF miniature resource grammar for Tswana: modelling the proper verb". In: *Language Resources and Evaluation* 51.1 (Mar. 2017), pp. 159–189.

[230] J. Proskurnia, M. Cartright, L. G. Pueyo, I. Krka, J. B. Wendt, T. Kaufmann, and B. Miklos. "Template Induction over Unstructured Email Corpora". In: *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*. Ed. by R. Barrett, R. Cummings, E. Agichtein, and E. Gabrilovich. ACM, 2017, pp. 1521–1530.

[231] R. Puduppully, L. Dong, and M. Lapata. "Data-to-Text Generation with Content Selection and Planning". In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, pp. 6908–6915.

[232] Y. Puzikov and I. Gurevych. “BinLin: A Simple Method of Dependency Tree Linearization”. In: *Proceedings of the First Workshop on Multilingual Surface Realisation*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 13–28.

[233] R. Rajkumar, D. Espinosa, and M. White. “The OSU System for Surface Realization at Generation Challenges 2011”. In: *ENLG 2011 - Proceedings of the 13th European Workshop on Natural Language Generation, 28-30 September 2011, Nancy, France*. Ed. by C. Gardent and K. Striegnitz. The Association for Computer Linguistics, 2011, pp. 236–238.

[234] A. Ramos-Soto, A. J. B. Diz, S. Barro, and J. Taboada. “Linguistic Descriptions for Automatic Generation of Textual Short-Term Weather Forecasts on Real Prediction Data”. In: *IEEE Transactions on Fuzzy Systems* 23.1 (2015), pp. 44–57.

[235] A. Ramos-Soto, J. J. Gallardo, and A. J. B. Diz. “Adapting SimpleNLG to Spanish”. In: *Proceedings of the 10th International Conference on Natural Language Generation, INLG 2017, Santiago de Compostela, Spain, September 4-7, 2017*. Association for Computational Linguistics, 2017, pp. 144–148.

[236] A. Ranta. *Grammatical framework: Programming with multilingual grammars*. CSLI Publications, Center for the Study of Language and Information Stanford, 2011.

[237] G. Recski, Á. Kovács, K. Gémes, J. Ács, and A. Kornai. “BME-TUW at SR’20: Lexical grammar induction for surface realization”. In: *Proceedings of the Third Workshop on Multilingual Surface Realisation*. Barcelona, Spain (Online): Association for Computational Linguistics, Dec. 2020, pp. 21–29.

[238] A. L. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, and C. Wroe. “OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns”. In: *Engineering Knowledge in the Age of the Semantic Web, 14th International Conference, EKAW 2004, Whittlebury Hall, UK, October 5-8, 2004, Proceedings*. Ed. by E. Motta, N. Shadbolt, A. Stutt, and N. Gibbins. Vol. 3257. Lecture Notes in Computer Science. Springer, 2004, pp. 63–81.

[239] R. Reddy. *Speech understanding systems: summary of results of the five-year research effort at Carnegie-Mellon University*. Tech. rep. Department of Computer Science, Carnegie-Mellon University, 1977.

[240] M. Reid, J. Hu, G. Neubig, and Y. Matsuo. “AfroMT: Pretraining Strategies and Reproducible Benchmarks for Translation of 8 African Languages”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*. Ed. by M. Moens, X. Huang, L. Specia, and S. W. Yih. Association for Computational Linguistics, 2021, pp. 1306–1320.

[241] E. Reiter. “Has a Consensus NL Generation Architecture Appeared, and is it Psycholinguistically Plausible?” In: *Proceedings of the Seventh International Workshop on Natural Language Generation, INLG 1994, Kennebunkport, Maine, USA, June 21-24, 1994*. Association for Computational Linguistics, 1994.

[242] E. Reiter. “An Architecture for Data-to-Text Systems”. In: *Proceedings of the Eleventh European Workshop on Natural Language Generation, ENLG 2007, Schloss Dagstuhl, Germany, June 17-20, 2007*. Ed. by S. Busemann. 2007.

[243] E. Reiter. *Hallucination in Neural NLG*. `https://ehudreiter.com/2018/11/12/hallucination-in-neural-nlg/`. Accessed: 2022-06-13.

[244] E. Reiter and S. Sripada. “Should Corpora Texts Be Gold Standards for NLG?” In: *Proceedings of the International Natural Language Generation Conference, Harriman, New York, USA, July 2002*. Association for Computational Linguistics, 2002, pp. 97–104.

[245] Y. Ren, A. Parvizi, C. Mellish, J. Z. Pan, K. van Deemter, and R. Stevens. “Towards Competency Question-Driven Ontology Authoring”. In: *The Semantic Web: Trends and Challenges*. Ed. by V. Presutti, C. d’Amato, F. Gandon, M. d’Aquin, S. Staab, and A. Tordai. Cham: Springer International Publishing, 2014, pp. 752–767.

[246] D. Sadoun, S. Mkhitaryan, D. Nouvel, and M. Valette. “ReadME generation from an OWL ontology describing NLP tools”. In: *Proceedings of the 2nd International Workshop on Natural Language Generation and the Semantic Web, WebNLG 2016, Edinburgh, UK, September 6, 2016*. Association for Computational Linguistics, 2016, pp. 46–49.

[247] Y. Safovich and A. Azaria. “Fiction Sentence Expansion and Enhancement via Focused Objective and Novelty Curve Sampling”. In: *32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9-11, 2020*. IEEE, 2020, pp. 835–843.

[248] R. C. A. Samuelson. *Zulu grammar*. Durban: Knox, 1930.

[249] L. Sanby, I. Todd, and C. M. Keet. “Comparing the Template-Based Approach to GF: the case of Afrikaans”. In: *Proceedings of the 2nd International Workshop on Natural Language Generation and the Semantic Web, WebNLG 2016, Edinburgh, UK, September 6, 2016*. Association for Computational Linguistics, 2016, pp. 50–53.

[250] B. Sands. “Africa’s linguistic diversity”. In: *Language and Linguistics Compass* 3.2 (2009), pp. 559–580.

[251] A. Shimorina and C. Gardent. “Surface Realisation Using Full Delexicalisation”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Ed. by K. Inui, J. Jiang, V. Ng, and X. Wan. Association for Computational Linguistics, 2019, pp. 3084–3094.

[252] G. Sibanda. “Vowel processes in Nguni: Resolving the problem of unacceptable VV sequences”. In: *Selected Proceedings of the 38th Annual Conference on African Linguistics, Gainesville, Florida, March 22-25, 2007*. Ed. by M. Matondo, F. M. Laughlin, and E. Potsdam. Cascadilla Proceedings Project, Somerville, Massachusetts, USA, 2007, pp. 38–55.

[253] N. E. Sigcau. “Educational implications of nonstandard varieties of Xhosa”. MA thesis. University of Cape Town, 1998.

[254] E. Simperl, M. Mochol, and T. Bürger. “Achieving Maturity: the State of Practice in Ontology Engineering in 2009”. In: *International Journal of Computer Science and Applications* 7.1 (2010), pp. 45–65.

[255] K. D. Smedt, H. Horacek, and M. Zock. “Architectures for Natural Language Generation: Problems and Perspectives”. In: *Trends in Natural Language Generation, An Artificial Intelligence Perspective, Fourth European Workshop, EWNLG ’93, Pisa, Italy, April 28-30, 1993, Selected Papers*. Ed. by G. Adorni and M. Zock. Vol. 1036. Lecture Notes in Computer Science. Springer, 1993, pp. 17–46.

[256] M. A. Sobrevilla Cabezudo and T. Pardo. “NILC-SWORNEMO at the Surface Realization Shared Task: Exploring Syntax-Based Word Ordering using Neural Models”. In: *Proceedings of the First Workshop on Multilingual Surface Realisation*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 58–64.

[257] M. A. Sobrevilla Cabezudo and T. Pardo. “NILC at SR’20: Exploring Pre-Trained Models in Surface Realisation”. In: *Proceedings of the Third Workshop on Multilingual Surface Realisation*. Barcelona, Spain (Online): Association for Computational Linguistics, Dec. 2020, pp. 50–56.

[258] S. Spiegler, A. van der Spuy, and P. A. Flach. “Ukwabelana - An Open-Source Morphological Zulu Corpus”. In: *Proceedings of the 23rd International Conference on Computational Linguistics, 23-27 August 2010, Beijing, China*. 2010, pp. 1020–1028.

[259] R. E. Sripada S.G. and I. Davy. “SUMTIME-MOUSAM: Configurable Marine Weather Forecast Generator”. In: *Expert Update* 6.3 (2003), pp. 4–10.

[260] S. Sripada, N. Burnett, R. Turner, J. Mastin, and D. Evans. “A Case Study: NLG meeting Weather Industry Demand for Quality and Quantity of Textual Weather Forecasts”. In: *Proceedings of the Eighth International Natural Language Generation Conference, Including Proceedings of the INLG and SIGDIAL 2014 Joint Session, 19-21 June 2014, Philadelphia, PA, USA*. ACL, 2014, pp. 1–5.

[261] Statistics South Africa. *Census 2011 : Census in brief*. `http://www.statssa.gov.za/census/census_2011/census_products/Census_2011_Census_in_brief.pdf`. Accessed: 23 November 2017. 2012.

[262] Statistics South Africa. *South African Census Community Profiles 2011*. DataFirst, 2015.

[263] H. Stenzhorn. “XtraGen - A Natural Language Generation System Using XML and Java-Technologies”. In: *The 2nd Workshop on NLP and XML, NLPXML@COLING 2002, Taipei, Taiwan, August 24 - September 1, 2002*. 2002.

[264] R. Stevens, J. Malone, S. Williams, R. Power, and A. Third. “Automating generation of textual class definitions from OWL to English”. In: *Journal of Biomedical Semantics* 2.S-2 (2011), S5.

[265] A. Sudhakar, B. Upadhyay, and A. Maheswaran. “”Transforming” Delete, Retrieve, Generate Approach for Controlled Text Style Transfer”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Ed. by K. Inui, J. Jiang, V. Ng, and X. Wan. Association for Computational Linguistics, 2019, pp. 3267–3277.

[266] I. Sutskever, O. Vinyals, and Q. V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. 2014, pp. 3104–3112.

[267] E. Teich. *Systemic functional grammar in natural language generation*. Ed. by R. Fawcett. Communication in artificial intelligence. London: Cassell, 1999.

[268] H. Thompson. “Strategy and tactics: A model for language production”. In: *Papers from the 13th Regional Meeting of the Chicago Linguistics Society, Chicago, Illinois, USA*. Ed. by W. Beach, S. Fox, and S. Philsoph. Vol. 13. 1977, pp. 651–668.

[269] B. D. Trisedya, J. Qi, and R. Zhang. “Sentence Generation for Entity Description with Content-Plan Attention”. In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2020, pp. 9057–9064.

[270] D. Tsarkov and I. Horrocks. “FaCT++ Description Logic Reasoner: System Description”. In: *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*. Ed. by U. Furbach and N. Shankar. Vol. 4130. Lecture Notes in Computer Science. Springer, 2006, pp. 292–297.

[271] R. Turner, S. Sripada, E. Reiter, and I. P. Davy. “Generating Spatio-Temporal Descriptions in Pollen Forecasts”. In: *Proceedings of the 11st Conference of the European Chapter of the Association for Computational Linguistics, EACL 2006, April 3-7, 2006, Trento, Italy*. Ed. by D. McCarthy and S. Wintner. Association for Computer Linguistics, 2006.

[272] K. Upasani, D. King, J. Rao, A. Balakrishnan, and M. White. "The OSU/-Facebook Realizer for SRST 2019: Seq2Seq Inflection and Serialized Tree2Tree Linearization". In: *Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR 2019)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 68–74.

[273] K. van Deemter, M. Theune, and E. Krahmer. "Real versus Template-Based Natural Language Generation: A False Opposition?" In: *Computational Linguistics* 31.1 (2005), pp. 15–24.

[274] C. van der Lee, E. Krahmer, and S. Wubben. "PASS: A Dutch Data-to-Text System for Soccer, Targeted towards Specific Audiences". In: *Proceedings of the 10th International Conference on Natural Language Generation, INLG 2017, Santiago de Compostela, Spain, September 4-7, 2017*. Association for Computational Linguistics, 2017, pp. 95–104.

[275] A. Van der Spuy. "Bilabial Palatalisation in Zulu: A morphologically conditioned phenomenon". In: *Stellenbosch Papers in Linguistics Plus* 44 (2014), pp. 71–87.

[276] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett. 2017, pp. 5998–6008.

[277] P.-L. Vaudry and G. Lapalme. "Adapting SimpleNLG for Bilingual English-French Realisation". In: *Proceedings of the 14th European Workshop on Natural Language Generation, ENLG 2013, August 8-9, 2013, Sofia, Bulgaria*. 2013, pp. 183–187.

[278] H. Verhoef and A. Van Eeden. "Identifying the challenges of creating an optimal dissemination geography for census". In: *South African Journal of Geomatics* 4.1 (2015), pp. 50–64.

[279] R. L. Walls, J. Deck, R. Guralnick, S. Baskauf, R. Beaman, S. Blum, S. Bowers, P. L. Buttigieg, N. Davies, D. Endresen, M. A. Gandolfo, R. Hanner, A. Janning, L. Krishtalka, A. Matsunaga, P. Midford, N. Morrison, É. Ó. Tuama, M. Schildhauer, B. Smith, B. J. Stucky, A. Thomer, J. Wieczorek, J. Whitacre, and J. Wooley. "Semantics in Support of Biodiversity Knowledge Discovery: An Introduction to the Biological Collections Ontology and Related Ontologies". In: *PLOS ONE* 9.3 (Mar. 2014), pp. 1–13.

[280] M. J. Weal, H. Alani, S. Kim, P. H. Lewis, D. E. Millard, P. A. S. Sinclair, D. D. Roure, and N. R. Shadbolt. "Ontologies as facilitators for repurposing web documents". In: *International Journal of Humman-Computuer Studies* 65.6 (2007), pp. 537–562.

[281] R. Weißgraeber and A. Madsack. “A working, non-trivial, topically indifferent NLG System for 17 languages”. In: *Proceedings of the 10th International Conference on Natural Language Generation*. Santiago de Compostela, Spain: Association for Computational Linguistics, Sept. 2017, pp. 156–157.

[282] W. Welmers. *African language structures*. University of California Press, 1973.

[283] M. White. “CCG Chart Realization from Disjunctive Inputs”. In: *Proceedings of the Fourth International Natural Language Generation Conference, INLG 2006, July 15-16, 2006, Sydney, Australia*. Ed. by N. Colineau, C. Paris, S. Wan, R. Dale, and A. Belz. Association for Computer Linguistics, 2006, pp. 12–19.

[284] M. White and T. Caldwell. “EXEMPLARS: A Practical, Extensible Framework For Dynamic Text Generation”. In: *Proceedings of the Ninth International Workshop on Natural Language Generation, INLG 1998, Niagara-on-the-Lake, Ontario, Canada, August 5-7, 1998*. 1998.

[285] M. White, R. Rajkumar, and S. Martin. “Towards broad coverage surface realization with CCG”. In: *Proceedings of the Workshop on Using Corpora for NLG: Language Generation and Machine Translation (UCNLG+ MT)*. 2007, pp. 267–276.

[286] G. Wilcock. “Pipelines, Templates and Transformations: XML for Natural Language Generation”. In: *Proceedings of the first NLP and XML Workshop; Workshop session of the 6th Natural Language Processing Pacific Rim Symposium, November 27-30, 2001, Hitotsubashi Memorial Hall, National Center of Sciences, Tokyo, Japan*. 2001.

[287] S. Wiseman, S. M. Shieber, and A. M. Rush. “Challenges in Data-to-Document Generation”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*. Ed. by M. Palmer, R. Hwa, and S. Riedel. Association for Computational Linguistics, 2017, pp. 2253–2263.

[288] S. Wiseman, S. M. Shieber, and A. M. Rush. “Learning Neural Templates for Text Generation”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*. Ed. by E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii. Association for Computational Linguistics, 2018, pp. 3174–3187.

[289] Y. W. Wong. “Learning for Semantic Parsing and Natural Language Generation Using Statistical Machine Translation Techniques”. PhD thesis. Department of Computer Sciences, University of Texas at Austin, Texas, 2007.

[290] X. Wu, T. Zhang, L. Zang, J. Han, and S. Hu. “Mask and Infill: Applying Masked Language Model for Sentiment Transfer”. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. Ed. by S. Kraus. ijcai.org, 2019, pp. 5271–5277.

[291] R. Yermakov, N. Drago, and A. Ziletti. “Biomedical Data-to-Text Generation via Fine-Tuning Transformers”. In: *Proceedings of the 14th International Conference on Natural Language Generation, INLG 2021, Aberdeen, Scotland, UK, 20-24 September, 2021*. Ed. by A. Belz, A. Fan, E. Reiter, and Y. Sripada. Association for Computational Linguistics, 2021, pp. 364–370.

[292] X. Yu, A. Falenska, M. Haid, N. T. Vu, and J. Kuhn. “IMSurReal: IMS at the Surface Realization Shared Task 2019”. In: *Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR 2019)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 50–58.

[293] X. Yu, S. Tannert, N. T. Vu, and J. Kuhn. “IMSurReal Too: IMS in the Surface Realization Shared Task 2020”. In: *Proceedings of the Third Workshop on Multilingual Surface Realisation*. Barcelona, Spain (Online): Association for Computational Linguistics, Dec. 2020, pp. 35–41.

[294] B. Zawada and M. N. Ngcobo. “A cognitive and corpus-linguistic re-analysis of the acquisition of the Zulu noun class system”. In: *Language Matters* 39.2 (2008), pp. 316–331.

[295] C. Zhao, M. A. Walker, and S. Chaturvedi. “Bridging the Structural Gap Between Encoding and Decoding for Data-To-Text Generation”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*. Ed. by D. Jurafsky, J. Chai, N. Schluter, and J. R. Tetreault. Association for Computational Linguistics, 2020, pp. 2481–2491.